Université de Montréal

VGCN-BERT : Augmenting BERT with Graph Embedding for Text Classification – Application to offensive language detection

par

Zhibin Lu

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

17 juin 2020

Université de Montréal

Faculté des études supérieures et postdoctorales

Ce mémoire intitulé

VGCN-BERT : Augmenting BERT with Graph Embedding for Text Classification – Application to offensive language detection

présenté par

Zhibin Lu

a été évalué par un jury composé des personnes suivantes :

Fabian Bastin
(président-rapporteur)

Jian-Yun Nie
(directeur de recherche)

Philippe Langlais
(membre du jury)

Résumé

Le discours haineux est un problème sérieux sur les média sociaux. Dans ce mémoire, nous étudions le problème de détection automatique du langage haineux sur réseaux sociaux. Nous traitons ce problème comme un problème de classification de textes.

La classification de textes a fait un grand progrès ces dernières années grâce aux techniques d'apprentissage profond. En particulier, les modèles utilisant un mécanisme d'attention tel que BERT se sont révélés capables de capturer les informations contextuelles contenues dans une phrase ou un texte. Cependant, leur capacité à saisir l'information globale sur le vocabulaire d'une langue dans une application spécifique est plus limitée.

Récemment, un nouveau type de réseau de neurones, appelé Réseau de convolution de graphe (GCN), émerge. Il intègre les informations des voisins en manipulant un graphique global pour prendre en compte les informations globales, et il a obtenu de bons résultats dans de nombreuses tâches, y compris la classification de textes.

Par conséquent, notre motivation dans ce mémoire est de concevoir une méthode qui peut combiner à la fois les avantages du modèle BERT, qui excelle en capturant des informations locales, et le modèle GCN, qui fournit les informations globale du langage.

Néanmoins, le GCN traditionnel est un modèle d'apprentissage transductif, qui effectue une opération convolutionnelle sur un graphe composé d'éléments à traiter dans les tâches (c'est-à-dire un graphe de documents) et ne peut pas être appliqué à un nouveau document qui ne fait pas partie du graphe pendant l'entraînement. Dans ce mémoire, nous proposons d'abord un nouveau modèle GCN de vocabulaire (VGCN), qui transforme la convolution au niveau du document du modèle GCN traditionnel en convolution au niveau du mot en utilisant les co-occurrences de mots. En ce faisant, nous transformons le mode d'apprentissage transductif en mode inductif, qui peut être appliqué à un nouveau document.

Ensuite, nous proposons le modèle Interactive-VGCN-BERT qui combine notre modèle VGCN avec BERT. Dans ce modèle, les informations locales captées par BERT sont combinées avec les informations globales captées par VGCN. De plus, les informations locales et les informations globales interagissent à travers différentes couches de BERT, ce qui leur permet d'influencer mutuellement et de construire ensemble une représentation finale pour la classification. Via ces interactions, les informations de langue globales peuvent aider à

distinguer des mots ambigus ou à comprendre des expressions peu claires, améliorant ainsi les performances des tâches de classification de textes.

Pour évaluer l'efficacité de notre modèle Interactive-VGCN-BERT, nous menons des expériences sur plusieurs ensembles de données de différents types – non seulement sur le langage haineux, mais aussi sur la détection de grammaticalité et les commentaires sur les films. Les résultats expérimentaux montrent que le modèle Interactive-VGCN-BERT surpasse tous les autres modèles tels que Vanilla-VGCN-BERT, BERT, Bi-LSTM, MLP, GCN et ainsi de suite. En particulier, nous observons que VGCN peut effectivement fournir des informations utiles pour aider à comprendre un texte haiteux implicit quand il est intégré avec BERT, ce qui vérifie notre intuition au début de cette étude.

Mots clés : Classification de textes, BERT, Réseau de convolution de graphe, Détection automatique du langage haineux.

Abstract

Hate speech is a serious problem on social media. In this thesis, we investigate the problem of automatic detection of hate speech on social media. We cast it as a text classification problem.

With the development of deep learning, text classification has made great progress in recent years. In particular, models using attention mechanism such as BERT have shown great capability of capturing the local contextual information within a sentence or document. Although local connections between words in the sentence can be captured, their ability of capturing certain application-dependent global information and long-range semantic dependency is limited.

Recently, a new type of neural network, called the Graph Convolutional Network (GCN), has attracted much attention. It provides an effective mechanism to take into account the global information via the convolutional operation on a global graph and has achieved good results in many tasks including text classification.

In this thesis, we propose a method that can combine both advantages of BERT model, which is excellent at exploiting the local information from a text, and the GCN model, which provides the application-dependent global language information.

However, the traditional GCN is a transductive learning model, which performs a convolutional operation on a graph composed of task entities (i.e. documents graph) and cannot be applied directly to a new document. In this thesis, we first propose a novel Vocabulary GCN model (VGCN), which transforms the document-level convolution of the traditional GCN model to word-level convolution using a word graph created from word co-occurrences. In this way, we change the training method of GCN, from the transductive learning mode to the inductive learning mode, that can be applied to new documents.

Secondly, we propose an Interactive-VGCN-BERT model that combines our VGCN model with BERT. In this model, local information including dependencies between words in a sentence, can be captured by BERT, while the global information reflecting the relations between words in a language (e.g. related words) can be captured by VGCN. In addition, local information and global information can interact through different layers of BERT, allowing them to influence mutually and to build together a final representation for classification. In so

doing, the global language information can help distinguish ambiguous words or understand unclear expressions, thereby improving the performance of text classification tasks.

To evaluate the effectiveness of our Interactive-VGCN-BERT model, we conduct experiments on several datasets of different types – hate language detection, as well as movie review and grammaticality, and compare them with several state-of-the-art baseline models. Experimental results show that our Interactive-VGCN-BERT outperforms all other models such as Vanilla-VGCN-BERT, BERT, Bi-LSTM, MLP, GCN, and so on. In particular, we have found that VGCN can indeed help understand a text when it is integrated with BERT, confirming our intuition to combine the two mechanisms.

Keywords: Text classification, BERT, Graph convolutional network, Offensive language detection.

Contents

List of Tables

List of Figures

List of Abbreviations

NLP	*Natural Language Processing*
BoW	*Bag-of-Words*
TF-IDF	*Term frequency – Inverse Document frequency*
PCA	*Principal Component Analysis*
KNN	*K-nearest Neighbor*
SVM	*Support Vector Machine*
GloVe	*Global Vectors for Word Representation*
DNN	*Deep Neural Network*
HAN	*Hierarchical Attention Network*
GNN	*Graph Neural Network*
RNN	*Recurrent Neural Network*

CNN	*Convolutional Neural Network*
LSTM	*Long Short-Term Memory*
Bi-LSTM	*Bi-directional Long Short-Term Memory*
BERT	*Bidirectional Encoder Representations from Transformers*
GCN	*Graph Convolutional Networkds*
VGCN	*Vocabulary Graph Convolutional Networks*
SGD	*Stochastic Gradient Descent*
Adam	*Adaptive Moment Estimation*

Acknowledgements

I am grateful to my advisor Jian-Yun Nie. It seems that I am restless, I thank him for his tolerance and help. His insight and competence to the research has given me great help. I thank him for review my paper all the night before the deadline so that the paper "VGCN-BERT: Augmenting BERT with Graph Embedding for Text Classification" can be published.

At the same time, I thank my family, my beloved wife and my lovely son, for their understanding and support so that I can complete my studies.

Chapter 1

INTRODUCTION

1.1. Problem of Offensive Language Detection

Humans have entered a new era of information explosion, and the Internet is an information-based ecosystem that contains an enormous amount of information. End-users use Internet to transmit all kinds of information through different platforms. Not all the information is of interest to a given user. Text classification is a crucial tool to help select the information useful to an end-user. Text classification is a particular task in Natural Language Processing (NLP), which aims to classify a text into several categories according to the content of the text. It can help people organize texts into different categories and filter useful text information. The applications of text classification include spam filtering, categorization of news articles, detection of hate speech and offensive language, etc. In this thesis, we focus on the problem of the detection of hate speech and offensive language.

In recent years, an increasing number of users are subject to offensive languages or have witnessed abusive and hateful texts online, which are related to sexism, racism, or other types of aggressive behaviors and cyberbullying. Governments have started to enact laws to prevent such behaviors, and major social platforms such as Facebook and Twitter are also censoring offensive posts with the assistance of artificial intelligence technologies, human reviewing, user reporting, and so on. However, the problem is still far from being fully resolved due to the complexity of the task.

In general, text classification approaches can be applied to the detection of hate speech and offensive languages. In the past few years, many text classification approaches have been successfully applied to offensive language detection [**4, 18, 1, 3**]. In this thesis, we also cast the problem as a classification problem. Depending on the content, a text is classified into categories, such as offensive/non-offensive.

1.2. Outlook of Our Solution

More generally, text classification is a fundamental problem in natural language processing (NLP) and has been extensively studied in many real applications. In recent years, we witnessed the emergence of text classification models based on neural networks such as convolutional neural networks (CNN) [**34**], recurrent neural networks (RNN) [**23**], and various models based on attention [**67**]. BERT [**16**] is one of the self-attention models that uses multi-task pre-training technique based on large corpora. It often achieves excellent performance, compared to CNN/RNN models and traditional models such as SVM, in many tasks [**16**] such as Named-entity Recognition (NER), text classification and reading comprehension.

The deep learning models excel by embedding both semantic and syntactic information in a learned representation. However, most of them are known to be limited in encoding long-range dependency information of the text [**5**]. The utilization of self-attention helps alleviate this problem, but the problem still remains. The problem stems from the fact that the representation is generated from a sentence or a document only, without taking into account explicitly the knowledge about the language (vocabulary). As a consequence, the immediate context of a word can be encoded into the representation of the word or of the sentence. for example, when encoding the sequence "text reading", one can incorporate the relation between "text" and "reading", This type of relation is called syntagmatic. However, the encoding will be limited in detecting and incorporating the relation between "reading" and "comprehension" or "understanding". This second type of relation is called paradigmatic. It s true that through the analysis of a large set of texts, word embedding can incorporate implicitly some of the paradigmatic relations by creating similar embeddings for semantically related words [**51**]. However, no explicit paradigmatic relation is incorporated into the classification process. Intuitively, our understanding of a text uses both types of relations. This issue also exists in the task of offensive language detection. Let us consider the following two examples from the movie review dataset and the offensive language dataset, respectively:

— *"Although it's a bit smug and repetitive, this documentary engages your brain in a way few current films do."*

— *"You don't need a perverted religion to feel compassion for humanity."*

For the first sentence of the movie review, both negative and positive opinions appear in the sentence. Yet the positive attitude *"a way few current films do"* expresses a very strong opinion of the innovative nature of the movie in an implicit way. Without connecting this expression more explicitly to the meaning of *"innovation"* in the context of movie review comments, the classifier may under-weigh this strong opinion and the sentence may be wrongly classified to be negative. For this example, it is important to leverage the connection between these expressions so that the classifier could understand it as a strong positive opinion. Local connections, e.g. based on self-attention that connects the expression to other tokens in the

sentence, may not help. For the second sentence of offensive language, it is also difficult for a machine to differentiate. First, it is a double negative sentence and will be misjudged as positive. Second, *"perverted"* is not a common offensive word, and it rarely appears in the corpus. Deep learning models even cannot find the word in their pre-trained vocabulary. Therefore, without relevant information connecting *"perverted religion"* to *"forcing imam"*, the sentence may also be wrongly classified to be non-offensive.

In recent studies, approaches have also been developed to take into account the global information between words and concepts. The most representative work is Graph Convolutional Networks (GCN) [**36**] and its variant Text GCN [**76**], in which words in a language are connected in a graph. By performing convolution operations on neighbor nodes in the graph, the representation of a word will incorporate those of the neighbors, allowing to integrate the global context of a domain-specific language to some extent. For example, the meaning of *"new"* can be related to that of *"innovation"* and *"surprised"* through the connections between them. However, GCNs that only take into account the global vocabulary information may fail to capture local information (such as word order), which is very important in understanding the meaning of a sentence. This is shown in the following examples, where the position of *"work"* in the sentence will change the meaning of the sentence:

— *"skip work to see it at the first opportunity."*

— *"skip to see it, work at the first opportunity."*

Without local information, GCNs are unable to make difference between them.

In this thesis, inspired by GCN [**36, 76**] and self-attention mechanism in BERT, we propose to combine the strengths of both mechanisms in the same model. We aim to produce a representation for a sentence containing both local information in the sentence and global information about the vocabulary. To this end, we first construct a graph convolutional network on the vocabulary graph based on the word co-occurrence information, which aims at encoding the global information of the language, then feed the graph embedding and document's word embedding together to a self-attention encoder in BERT. The word embedding and graph embedding then interact with each other through the self-attention mechanism while learning the classifier. This way, the classifier can not only make use of both local information and global information, but also allow them to guide each other via the attention mechanism so that the final representation built up for classification will integrate gradually both local and global information. We also expect that the connections between words in the initial vocabulary graph can be spread to more complex expressions in the sentence through the layers of self-attention.

For the earlier movie review example, we do not expect that an expression such as "*a way few current films do*" be incorporated in a vocabulary GCN. However, through different layers of interactions between local and global information, the meaning of "*a way few current films do*" can be successfully connected to that of *"innovation"* and

"supprised". Enhanced with the latter, the classifier is now able to make the right decision. Similarly, through different layers of interactions, the meaning of *"perverted religion"* can also be successfully connected to that of *"forcing imam"*, then make the correct decision.

The above approach is implemented in our model called VGCN-BERT. Comparing with existing GCN models which conduct convolution operations on document level, our VGCN performs the convolution operations at the word level, such that it can exploit a vocabulary graph which represents the global language information. In addition, the VGCN model constructed on the vocabulary can be applied to any document, i.e. is inductive, while the traditional GCNs are transductive, i.e. only applicable to the documents included in the graph during the training phase.

We carry out experiments on a different datasets, not only for hate speech and offensive language detection, but also on other text classification tasks: movie review and grammaticality. We show that our proposed method outperforms VGCN and BERT alone.

1.3. Contribution

The contribution of this work is as follows:

— *Vocabulary graph:* We propose a method to build a flexible vocabulary graph (VGraph) to model semantic correlation between words, which is able to incorporate word co-occurrence relations and extra semantic correlation.

— *Vocabulary GCN:* We propose a novel inductive graph neural network called vocabulary graph convolutional network (VGCN) for text representations learning using the vocabulary graph. Our experiments demonstrate that it is faster, more concise, and more efficient than the existing GCN models.

— *Combining global and local information*: There has not been much work trying to combine local information captured by BERT and global information of a language. We demonstrate that their combination is beneficial.

— *Interaction between local and global information through attention mechanism*: We propose to use the self-attention mechanism to make a tight integration of local information and global information, allowing them to interact through different layers of networks. Relations between words in the initial graph can be spread to other words and expressions in the sentence. Our experiments will show that this is helpful in classification tasks.

1.4. Organization

The remaining of the thesis is organized as follows:

Chapter 1 introduces the background of the text classification task and the problem of offensive language detection, then outlines our solution and our contributions.

Chapter 2 provides a review of some typical approaches for text classification and the basic deep learning methods that are related to our study.

Chapter 3 introduces our methods that integrate the background language information about the vocabulary with the local information of the document. We will describe the method of building the Vocabulary Graph, to perform Vocabulary GCN. We will describe our method to combine VGCN and BERT – Interactive-VGCN-BERT model.

Chapter 4 describes the experimental settings and results. We will show that our approach can outperform the existing state-of-the-art models.

Chapter 5 will conclude the work and give some possible avenues for future improvements.

Appendix describes the experimental settings and results for the full VGCN model on some other document collections in which documents are connected.

We would like to note that the work described in this thesis has been published at a recent conference – European Conference on Information Retrieval [**45**].

Chapter 2

LITERATURE REVIEW

In this chapter, we introduce the current state of research on the task of offensive language detection, as well as text classification approaches, which are closely related to our problem. We will cover traditional approaches based on statistical and machine learning methods, and approaches based on deep learning methods. we only introduce important methods and technologies in the field of text classification, and will further introduce approaches that are closely related to our proposed model.

2.1. Existing Approaches to Offensive Language Detection

In the past few years, many text classification approaches have been applied to hate speech and offensive language detection. Among them, some deep learning approaches [**4, 18, 1**] have been proposed. These studies have achieved very good results, compared to the traditional machine learning approaches. The advantage of deep learning methods comes from the fact that useful features can be automatically extracted through the training process. This can effectively solve the problem of limited features constructed manually.

Notice, however, that a recent study [**3**] points out some issues about the methodology in previous work. In particular, Many existing approaches use the complete dataset including the test data to extract features for training the models [**4**]. This does not correspond to a realistic situation: the test data is unknown during the training phase. The datasets used in the experiments of hate language detection also had some issues. One of the most widely used 16K dataset [**72**] has a strong user bias. 65% of the messages marked as hateful ("sexist" or "racist") were produced by only 2 users. This issue will cause the model tending to identify users rather than detect the offensive language. To avoid the issue, the author created a higher quality dataset by merging datasets of [**72**] and [**14**]. We will use this merged dataset as one of our test datasets, and will avoid these issues when designing our approach for this task.

2.2. Text Classification and Traditional Approaches

We will start with a description of the general text classification problem. This will lay a basis for the offensive language detection problem.

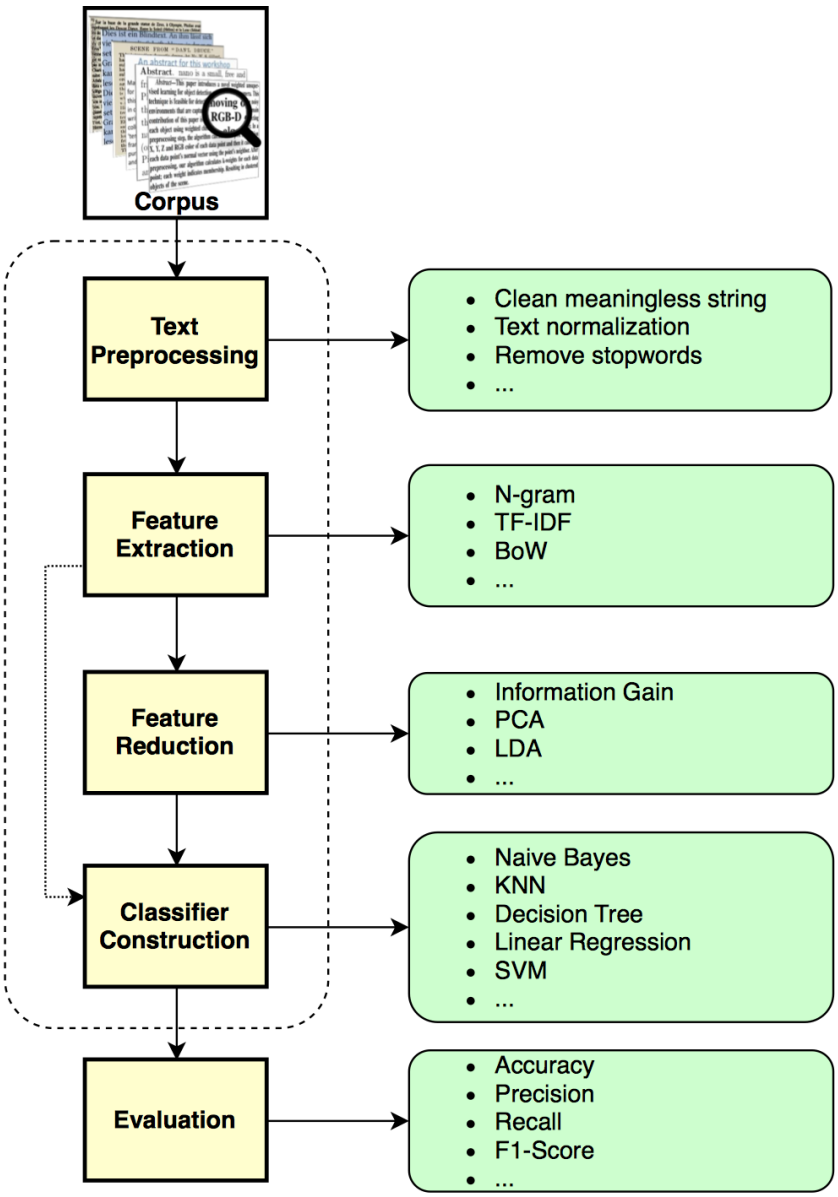


Figure 2.1. Framework of Text Classification based on Machine Learning Approaches

2.2.1. Framework of Text Classification Approach

Text classification usually relies on a set of pre-classified (labeled) texts to learn a classification model, which is then applied to new texts (test texts). Figure 2.1 shows the framework of a traditional text classification system.

A large number of studies show that automatic classification accuracy can become comparable to that of human classification, and can be applied to many domains without the intervention of domain experts. In Figure 2.1, we can see several typical processes in text classification: text preprocessing, feature extraction, feature reduction selection (optional), classifier construction, and evaluation. We will provide some details of these processes in the following sections.

2.2.2. Text Pre-processing

Text preprocessing aims to convert the original text to a set of useful elements (e.g. words, or word stems). In addition to text format conversion, text preprocessing typically involves removing elements that are not useful for classification. For example, some fields in a text collected from the Web (e.g. modification date) may not be useful for a given classification task. Punctuation symbols also are usually removed. Some functional words, called stopwords (e.g. "of", "the"), are also removed. The remaining words can go through a normalization process to map a word to a normalized form. For example, characters are usually lower-cased. Each word will undergo Spelling Correction, Slang and Abbreviation standardization, Stemming/Tokenization [**49, 25, 46, 56, 37, 63, 61**]. We take the detection of hate speech and offensive language as an example, consider the following tweet from the dataset of Founta et al. [**17**] :

"RT @sugalmond: Jang dongwoo what is the fucking matter w u https://t.co/UznoutSGmn"

After all the text pre-processing, it should become a list of tokens:

{"jang", "dongwoo", "what", "be", "the", "fuck", "matter", "you"}

2.2.3. Feature Extraction

In general, texts are unstructured data sets. However, in the traditional machine learning approaches, these unstructured text sequences are required to be converted into a feature space before using mathematical model for classification. The most common methods of feature extractions in traditional machine learning are BOW, N-Gram, and Term Frequency-Inverse Document Frequency (TF-IDF).

2.2.3.1. *Bag-of-words (BOW)*

The Bag-of-words model is the earliest effort to generate document representation in the NLP field. For a document, the BOW model represents a document as a collection of mentioned words (a bag of words) that appear in it, which ignores its word order and syntactic structure. During the processing, a dictionary is generated. The whole set of words form a representation space. In this method, words are independent from each other. For example, given two documents as follows:

— *"Bob likes to play basketball, Jim likes too."*

— *"Bob also likes to play football games."*

we can obtain the dictionary,

Dictionary = {1:"Bob", 2. "like", 3. "to", 4. "play", 5. "basketball", 6. "also", 7. "football", 8. "games", 9. "Jim", 10. "too"}

This dictionary contains a total of 10 different words. Using the index number of the dictionary, each document can be represented by a 10-dimensional vector with the index of the occurred word in the dictionary:

[1, 2, 1, 1, 1, 0, 0, 0, 1, 1]

[1, 1, 1, 1 ,0, 1, 1, 1, 0, 0]

The Bag-of-words model (BOW) is a reduced and simplified representation method, since the grammar and sequential information is ignored.

2.2.3.2. *Term Frequency-Inverse Document Frequency (TF-IDF)*

In the previous exemple, each word is weighted by tits frequency of occurrences in the text. Another commonly used weighting schema is IF-IDF. TF (Term Frequency) is the frequency of a term (word) in the text. IDF (Inverse Document Frequency) is a measure proposed by K. Sparck Jones [**32**], which aims at determining if a term is specific in a text and is not spread among many texts. The intuition is that a specific term for a text may represent a unique meaning for the text, thus should be weighted high. The common formula for TF-IDF is as follows, where the second element in the right part corresponds to IDF:

$$W(d_i,t_j) = TF(d_i, t_j) * log(\frac{N_d}{DF(t_j)}), \tag{2.2.1}$$

In this formula, N is the number of total documents and $DF(t)$ is the number of documents containing the term t in the corpus. Thus each document can be represented by a set of weighted terms.

2.2.4. Dimensionality Reduction

The feature extraction methods in the previous section can yield high-dimensional feature vector, leading to high time complexity and memory consumption for downstream processing. Therefore, the methods of feature dimension reduction are introduced to generate a low-dimensional feature vector without losing the extracted information. One method is Feature Selection, which selects the most effective features from the original feature word set to form a new feature vector. Another method is the spatial mapping method, which is to project high-dimensional feature vectors (generally linear mapping) to low dimensional space.

2.2.4.1. *Feature selection with Information Gain (IG)*

An important measure used in feature selection is Information Gain [**40**]. For a feature word, it is assumed that all categories have an average unconditional information entropy without knowing the feature word. Information gain (IG) measures the reduction of the entropy when the feature word is observed. It can be described as:

$$\begin{aligned} I(T,w) &= E(T) - E(T|w) \\ &= -\sum_{c\in C} Pr(T(d)=c)\log Pr(T(d)=c) \\ &+ \sum_{c\in C} Pr(T(d)=c, w=0)\log Pr(T(d)=c|w=0) \\ &+ \sum_{c\in C} Pr(T(d)=c, w=1)\log Pr(T(d)=c|w=1), \end{aligned} \tag{2.2.2}$$

where $E(T)$ is the average information entropy of all categories, $E(T|w)$ is the average information entropy of all categories under the condition of the feature word w appeared or not, $Pr(T(d)=c, w=0)$ is the probability of category c without feature word w, while $Pr(T(d)=c, w=1)$ is the probability of category c with feature word w. Once IG is calculated for each word, we can select k feature words with the largest IG values to form a new dictionary of features, and all documents are converted into k dimension feature vectors according to the new dictionary.

2.2.4.2. *Principal Component Analysis (PCA)*

Instead of selecting a subset of features from candidate terms, one can also transform the original features into a smaller set of composed features. Principal component analysis (PCA) is a popular method for this. PCA is a method to find a subspace in which the data approximately lies. It aims to find new variables that are uncorrelated and maximize the variance to "preserve as much variability as possible" [**31**].

Suppose a dataset $X \in \mathbb{R}^{n\times m}$, where m is the dimension and n is the size of the dataset, and $n \ll m$. The linear combination of jth features of X can be written as:

$$\sum_{j=1}^{m} w_j x_j = Xw, \tag{2.2.3}$$

where w is a vector of weight. The variance of this linear combination can be given as:

$$var(Xw) = w^T S w \tag{2.2.4}$$

where S is the covariance matrix of data. It aims to find the linear combination with the largest variance, that is to maximize $w^T S w - \lambda(w^T w - 1)$, where λ is a Lagrange multiplier. Once w is found,the initial representation of texts is then transformed to Xw.

2.2.5. Classifiers

A large number of classification algorithms have been proposed in the literature. We will not review all of them, but limit our review on KNN, Naive Bayes and SVM.

2.2.5.1. *K-nearest Neighbor (KNN)*

K-Nearest Neighbor (KNN) classifier is a classic classifier in traditional machine learning. In a typical KNN application [**28**], given a test document x, the KNN algorithm finds the k nearest neighbors of x among all the documents in the training set, and calculates the scores of all categories based the categories of the k neighbors. The similarity of x with each neighbor's document could also be taken into account. The general decision rule of KNN is:

$$\begin{aligned} f(x) &= argmax_j S(x,C_j) \\ &= \sum_{d_i \in KNN} Sim(x,d_i) y(d_i,C_j), \end{aligned} \tag{2.2.5}$$

where $S(x,C_j)$ is the score of candidate i for category j, $Sim(x,d_i)$ is the similarity function between a neighbor d_i and the new text x, and $y(d_i,C_j)$ is the (binary) class of class C_j for d_i. We assign the label $f(x)$ to the new document.

2.2.5.2. *Naive Bayes Classifier*

Another commonly used classifier is the Naive Bayes classifier, which assumes that the individual feature attributes of the document are independent of each other, based on which the posterior probabilities of the categories are calculated. The new test document is assigned to the category with the highest posterior probability.

According to the Bayesian equation, under the condition that the given document $d = (x_1,x_2,...,x_n)$ (for simplicity, the subscript of d is omitted, and x represents the ith feature of the document d), the probability of occurrence of the category $c_j (j = 1,2,...,m)$ (i.e, the Probability of the document d belongs to the category c_j) is:

$$\begin{aligned} p(c_j|d) &= \frac{p(c_j)p(x_1,x_2,...,x_n|c_j)}{p(x_1,x_2,...x_n)} \\ &= \frac{p(c_j)p(x_1|c_j)p(x_2|c_j)...p(x_n|c_j)}{p(x_1,x_2,...x_n)} \end{aligned} \tag{2.2.6}$$

After estimating the value of each probability in the equation by simple word frequency statistics, the posterior probability of each category $p(c_j|d)$ can be calculated.

2.2.5.3. *Support Vector Machine (SVM)*

The Support Vector Machine (SVM) method proposed by Vipnik et al. is based on statistical theory [**13**]. The basic idea of SVM is to find the optimal high-dimensional

classification hyperplane, which separates the two classes of data with the largest margin. SVM has received extensive attention in the field of machine learning. Thorsten Joachims is among the first to use SVM with the linear kernel function for text classification [**29**], and greatly improved the performance of text classification.

2.3. Deep Learning Approaches

Deep learning models have achieved state-of-the-art results across many domains, including a wide variety of NLP tasks as well as text classification.

In this section, we will introduce the new feature representation (i.e Word Embedding) method for Deep learning approaches, along with basic deep learning methods for text classification, such as Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Attention Networks, and Graph Neural Networks (GNNs).

2.3.1. Word Embedding

As aforementioned, the traditional feature extraction cannot directly account for the similarity between the words in the document since each word is used as independent features. Semantic relations between words cannot be captured. or example, *"astronaut"*, *"spaceman"*, *"cosmonaut"* and *"spationaut"* are often mentioned in the same context. However, their corresponding vector dimensions are orthogonal in the Bag-of-Words model. The inability to compare these similar words limits the performance of the BOW method. Additionally, the sequential information of the words in the sentence is not considered in the BOW representations. Word embedding can effectively solve these issues.

Bengio et al. proposed the statistical neural network language model [**6**], alleviating the dimensional disaster of the language model, and pioneered the concept of word embedding. It laid down the foundation for the subsequent study of word representation learning, such as Word2vec [**50**], GloVe [**55**] and so on. When word embeddings are used as input to a multi-layer perceptrons (MLP), one can often get better performance in text classification than the traditional feature extraction methods (such as TF-IDF) as input feature.

Word embedding is a feature learning technique in which each word or phrase from the vocabulary is mapped to a N dimensional vector of real numbers. Various word embedding methods have been proposed to translate unigrams into understandable input for machine learning algorithms. We will describe briefly Word2Vec, GloVe, FastText, and BERT, which have been successfully used for deep learning techniques.

2.3.1.1. *Word2Vec*

Mikolov et al. [**50**] presented "word to vector" representation as an improved word embedding architecture. The Word2Vec approach is the first widely used word embedding method,

which uses shallow neural networks with two hidden layers. Two different models are proposed: continuous bag-of-words (CBOW) and Skip-gram model, which try to maximize the prediction of word from its context words, or the reverse. Figure 2.2 shows a simple CBOW language model which tries to find the word based on surrounding words, and Skip-gram which tries to find surrounding words according to the word. For Skip-gram model, the goal is to maximize the following probability [**19**]:

$$\underset{\theta}{argmax} \prod_{w \in T} [\prod_{c \in c(w)} p(c|w;\theta)] \tag{2.3.1}$$

where w is a word, c its context (surrounding words), T refers to a text, and θ is the parameter of $p(c|w;\theta)$.

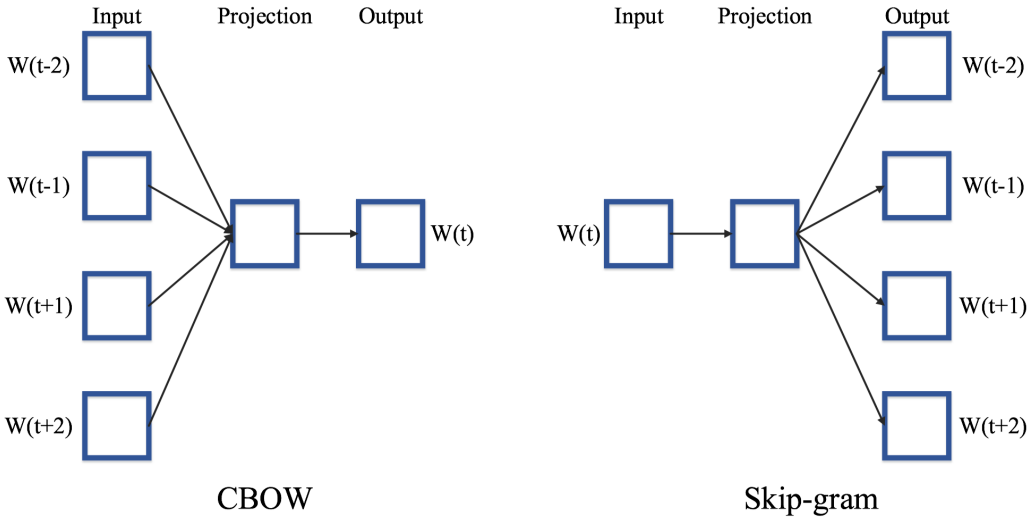


Figure 2.2. Schematic of the Continuous Bag-of-Words (CBOW) and the Skip-gram. Left is the CBOW structure that predicts the current word based on the context, and right is the Skip-gram that predicts surrounding words based on the given current word.

The learnable weights between the input layer and output layer represent $V \times N$ [**57**] as a matrix of W, where V is the vocabulary size and N is the hidden layer size. The following equations show how to calculate the word embedding (hidden layer h).

$$\begin{aligned} h_{Skip-gram} &= W^T x = W^T_{(k,.)} := v^T_{w_I}, \\ h_{CBOW} &= \frac{1}{C}(W^T(x_1 + x_2 + ... + x_C)^T \\ &= \frac{1}{C}(v_{w1} + v_{w1} + ... + v_{wC})^T, \end{aligned} \tag{2.3.2}$$

for the first equation, k is k-th row of W, v_{w_I} is the vector representation of the input word w_I, $h_{Skip-gram}$ essentially means copying the k-th row of W to h. For the second equation, C is the size of surrounding words, h_{CBOW} means calculate the average of the hidden layer of surrounding words.

Based on the co-occurrence of words, this method can exploit the text corpus tot extract the similarity between words, and generate representations to encode such similarity. For instance, the representations of two semantically similar words such as "woman" and "girl" can be close in the embedded space.

2.3.1.2. *Global Vectors for Word Representation (GloVe)*

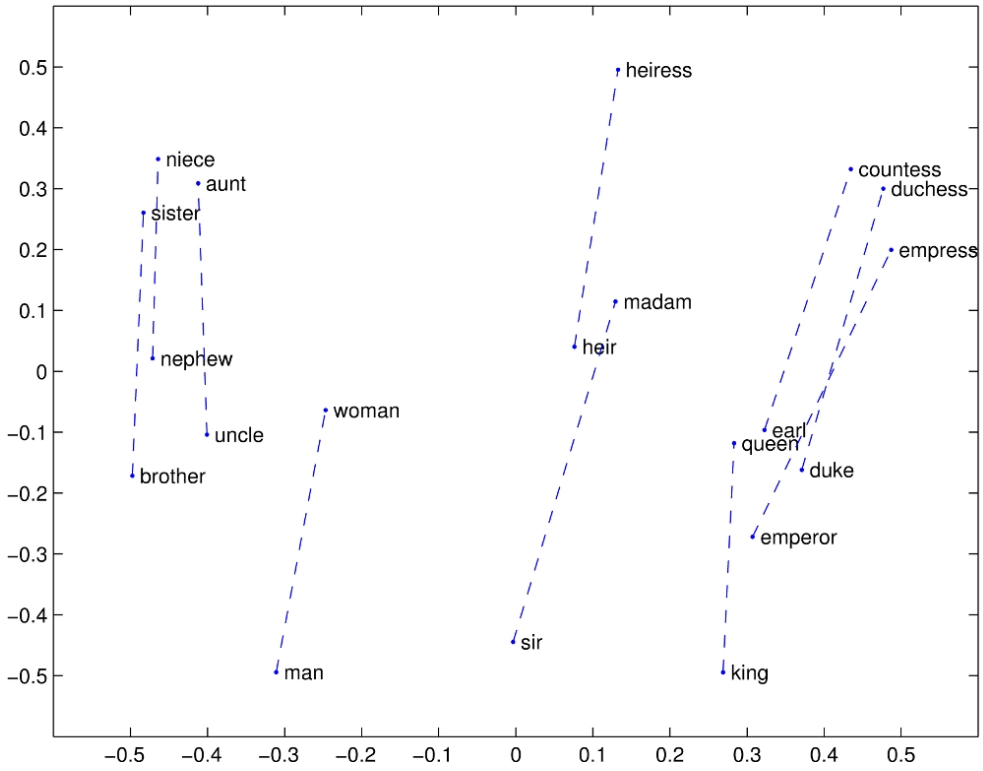


Figure 2.3. Visualization of the Global Vectors for Word Representation (GloVe).

Another powerful word embedding technique that has also been wildly used for text classification is Global Vectors (GloVe) [**55**]. The approach is very similar to the Word2Vec method, where each word is presented by a high dimension vector and trained based on the co-occurrence of the words. The pre-trained 50 dimensional word embedding used in many studies is based on 400,000 vocabularies trained over Wikipedia 2014 and Gigaword 5 as corpora. There were also other pre-trained word represenations by Glove in different dimensions, such as 100, 200, 300 dimensions. Figure 2.3 shows a visualization from [**47**] of the word distances over a sample data set using the same t-SNE technique. The objective equation is as follows:

$$f(w_i - w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{2.3.3}$$

where w_i refers to the word vector of word i, and $\tilde{w_k}$ refers to the k-th context word vector, and P_{ik} denotes to the probability of word k to occur in the context of word i. The above

equation reflects the idea that two words co-occuring often in the same contexts should be similar.

2.3.1.3. *FastText*

Joulin et al. proposed a simple and effective word embedding model [**33, 7**], named FastText. Instead of taking a whole word, it breaks it down into n-grams of characters. Each word, w, is represented as a bag of character n-gram. For example, given the word "classify" and $n = 3$, FastText will produce the following representation composed of character tri-grams:

<cl, cla, las, ass, ssi, sif, ify, fy>

Suppose we have a dictionary of n-grams of size G, and a word w which is associated as a representation vector z_g to each n-gram g. The equation for scoring the word w is [**7**]:

$$s(w, c) = \sum_{g \in g_w} z_g^T v_c \tag{2.3.4}$$

where $g_w \in \{1, 2, ..., G\}$ and v_c is the context representation vector for the word w.

The purpose of FastText is to accommodate the morphological variations of words, so that slightly different words (e.g. misspelled words) will result in similar embeddings. FastText is efficient with only one hidden layer and one output layer. On a standard multi-core CPU, it can train word embedding vectors from a billion word-level corpus in 10 minutes, and can classify more than 500,000 sentences with more than 300,000 categories in 1 minute. Facebook published pre-trained word embedding vectors of 300 dimensions for 294 languages which are trained on Wikipedia. On top of the n-grams encoding, FastText uses the Skip-gram language model [**7**] with default parameters.

2.3.2. Deep Neural Networks (DNNs)

DNN has been very successful in recent studies. DNN initially achieved great success in image and speech processing. It also achieved good results in text classification.

The basic deep learning architecture is multilayer perceptron (MLP), which is designed to learn by multi-connection of layers that every single layer only receives the connection from previous and provides connections only to the next layer in a hidden part [**38**], or it can be deemed as a stack of multi perceptrons. Figure 2.4 depicts the architecture of a standard DNN. In order to obtain non-linear fitting capabilities, the output of the hidden layer needs to go through a non-linear activation function, such as *Sigmoid* (Equation 2.3.5a), $ReLU$ [**52**]

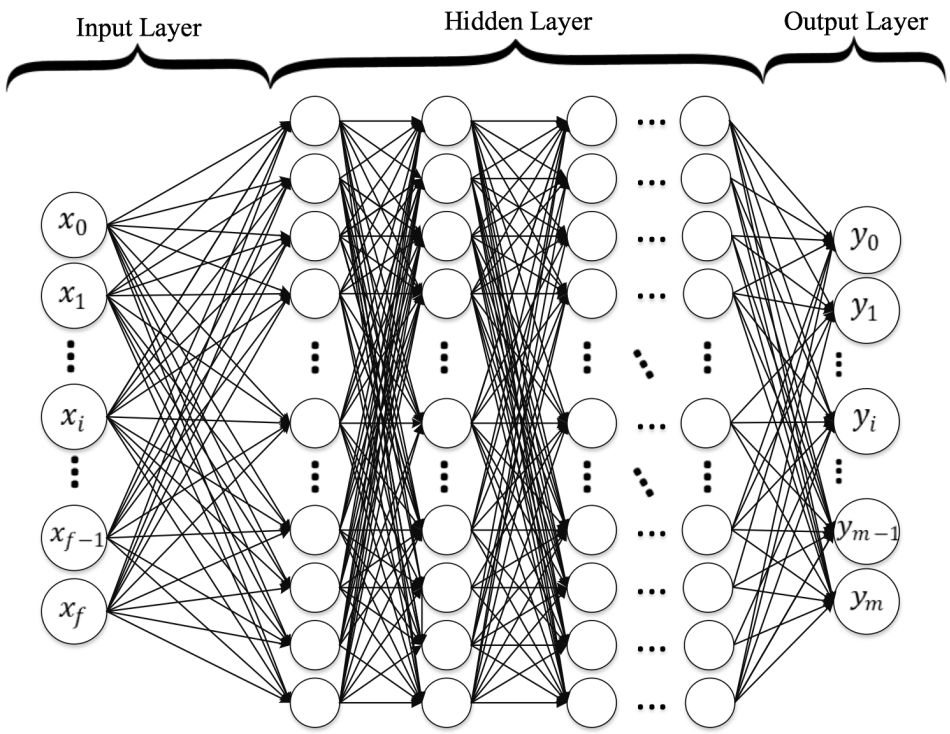


Figure 2.4. Schematic of a standard, fully connected deep neural network (DNN), also called multilayer perceptron (MLP).

(Equation 2.3.5b), *Gelu*, *Elu*, *Tanh* and so on [**53**].

$$f(x) = \frac{1}{1+e^{-x}} \in (0,1) \tag{2.3.5a}$$

$$f(x) = \max(0,x) \tag{2.3.5b}$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \forall j \in \{1,...,K\} \tag{2.3.5c}$$

The method takes as input the feature vector of text, which may be constructed via traditional feature extraction method such as TF-IDF (Section 2.2.3), or word embedding (Section 2.3.1), and connect with the first hidden layer of the DNN. The output layer is equal to the number of classes for multi-class classification or only one for binary classification. As with all deep learning models, the training of DNN generally has a standard process thatt is different from traditional statistical model. This usually includes: the $Softmax$ function (Equation 2.3.5c) to calculate the confidence of the different categories; a loss function (such as Cross-entropy loss, mean squared error (MSE)) to estimate the loss gap; and the standard back-propagation algorithm (e.g. SGD, Adam) to adjust the parameters of each layer.

DNN is equivalent to a super function with many parameters. Given a set of example pairs $(x, y), x \in X, y \in Y$, the goal is to learn the relationship between these input and

target spaces using hidden layers. In the text classification applications, the input is usually a string of words that is transformed into a vector representation.

2.3.3. Convolutional Neural Networks (CNNs)

A convolutional neural network (CNN) is a deep learning architecture that was originally applied to image processing. In NLP, CNN is commonly used for document classification [**69, 26, 35, 39**]. In a basic CNN for image processing, an image tensor is convolved with a set of kernels of size $d \times d$. The CNN convolution layers are called feature maps and can be stacked to provide multiple filters on the input. To reduce the computational complexity, CNN uses pooling to reduce the size of the output from one layer to the next in the network. Different pooling techniques are used to reduce outputs while preserving important features [**59**].

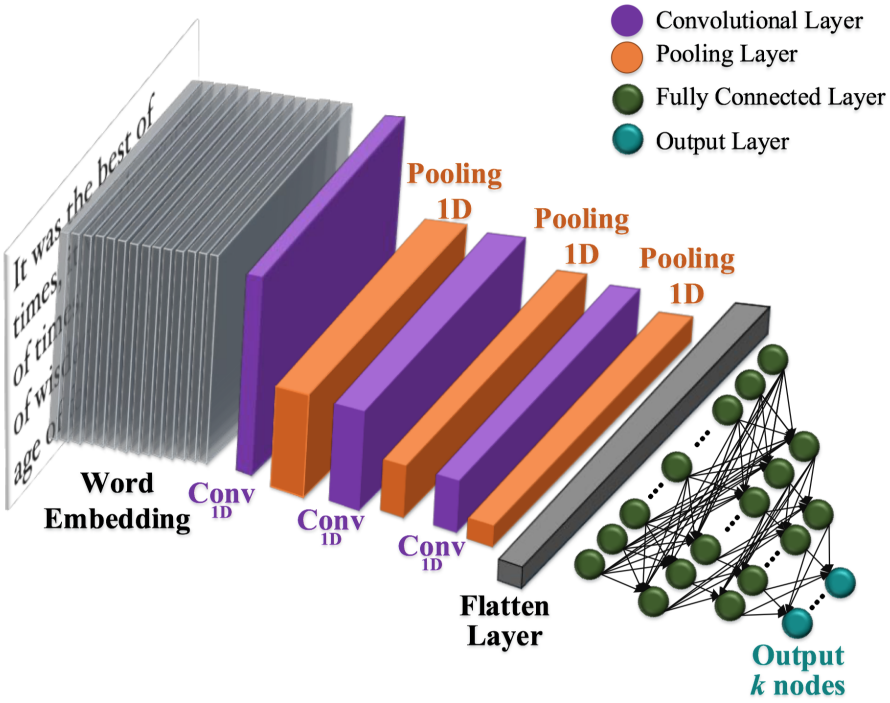


Figure 2.5. Convolutional neural network (CNN) architecture for text classification.

The most common pooling methods are max pooling and average pooling, which keep either the maximum element, or the average of all elements in the pooling window. To feed the pooled output from stacked featured maps to the next layer, the maps are flattened into one column. The final layers in a CNN are typically fully connected, e.g. a two-layer MLP. In general, during the back-propagation stage, both the weights and the feature detector filters are adjusted. A potential problem that arises when using CNN for text classification is the number of "channels" (i.e. the dimension of the feature space). While image classification applications generally have a few channels (e.g., only 3 input channels of RGB), the dimension may be very high (e.g., $20K$ words) for text classification applications [**30**]. Therefore, word

embeddings are used as input instead of words (one-hot representation). Figure 2.5 illustrates the CNN architecture for text classification which contains word embedding as input layer $1D$ convolutional layers, $1D$ pooling layer, fully connected layers, and output layer.

2.3.4. Recurrent Neural Networks (RNNs)

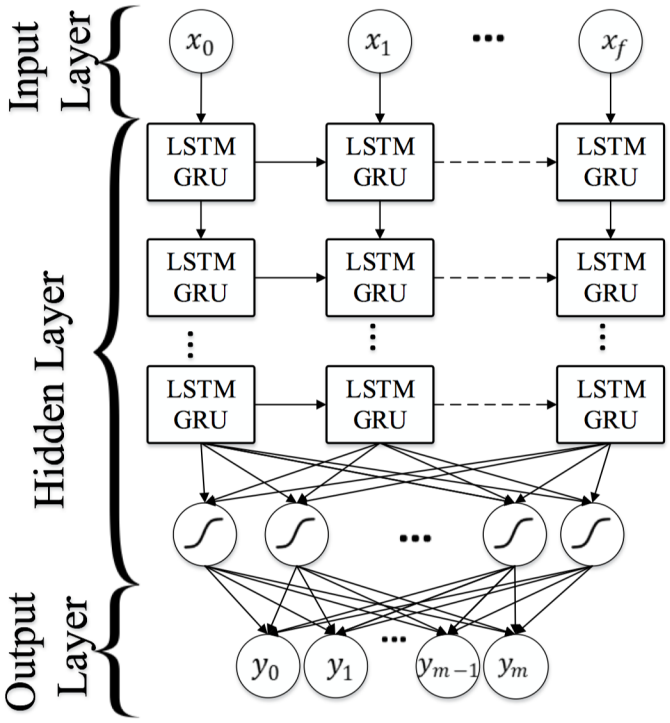


Figure 2.6. flowchart of Standard long short-term memory (LSTM)/Gated Recurrent Unit (GRU) recurrent neural networks.

Another deep learning architecture often used for text mining and classification is the recurrent neural network (RNN) [**64, 65**]. RNN is a powerful method for text, string, and sequential data classification because it specifically assigns a large number of weights to the previous nodes of a sequence. RNN uses a complex mechanism to consider the information of previous nodes, including the update of memory and the output of memory, which allows for better semantic analysis of texts. RNN methods such as LSTM and GRU are proved to be effective in text classification. A typical archtecure is shown in Figure 2.6, which contains the input layer (such as word embedding), hidden layers, and output layer. This method can be formulated as:

$$x_t = F(x_{t-1}, \vec{u_t}, \theta), \tag{2.3.6}$$

where x_t is the state at time t and u_t refers to the input at step t. More specifically, it can be parameterized by:

$$x_t = W_{rec}\sigma(x_{t-1}) + W_{in}\vec{u_t} + b \tag{2.3.7}$$

where W_{rec} rec refers to recurrent matrix weight, W_{in} in refers to input weights, b is the bias, and σ denotes an element-wise function.

2.3.5. Attention Networks

Attention is a mechanism proposed later. It achieves good performance in various NLP tasks.

2.3.5.1. *Attention Mechanism*

The attention mechanism [**75, 70**] is inspired by human eyes that focus on local information when observing things. The attention mechanism solves two issues of RNN model, that is non-parallel calculation and loss of long distance information because attention can be paid to any element at any position in the same sequence of words.

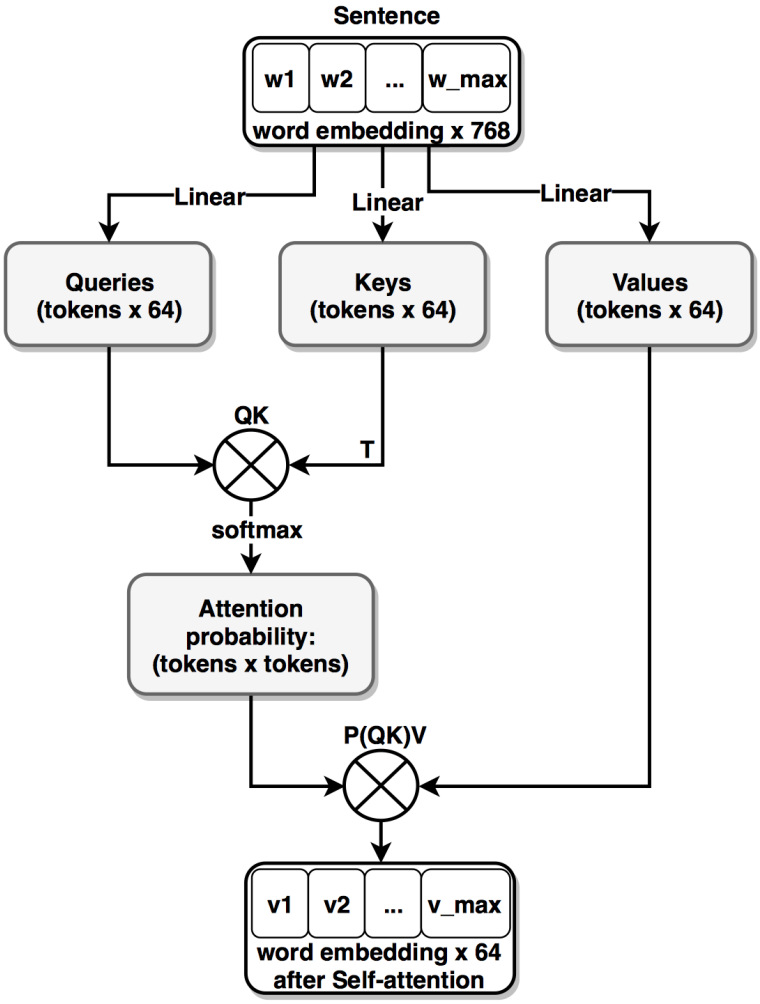


Figure 2.7. Flowchart of Self-attention. The number of 768 is an example of the dimension of word embedding vector, and 64 is an example of the dimension of word embedding after linear transformer.

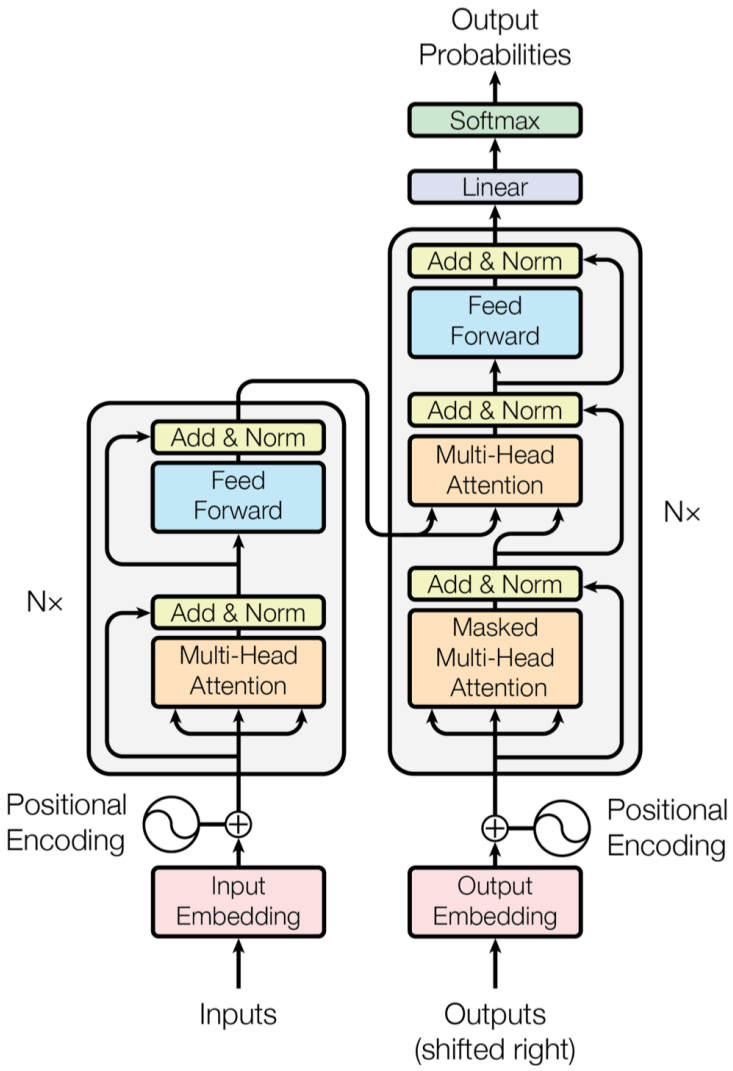


Figure 2.8. Architecture of Transformer component.

In NLP, the self-attention mechanism proposed by Vaswan et al. [**67**] is introduced and proved to greatly improve the performance in text classification tasks.

Self-attention: The flowchart of Self-attention is showed in figure 2.7. Firstly, the Query vectors, the Key vectors, and the Value vectors are generated respectively from linear transformations (multiplied by three weight matrices) based on the word embedding vectors. Note that the dimension of these new vectors is lower than the word embedding vectors, e.g. a 768 dimension word embedding vector is linearly transformed into a 64 dimension vector.

Secondly, for each word embedding ("query" - Q) in the input sentence, the scores for other words ("keys" - K, including itself) are calculated in turn, which are the dot product of the "query" vector and the "key" vector. That is, we will get a score matrix $S \in \mathbf{R}^{N\times N}$ where N is the length of tokens of the document. One row of the score matrix means the different weights of the other tokens to the token corresponding to the row index.

The third step is to calculate the attention probability. The formula for calculating the attention probability is as follows:

$$P_{attention}(Q,K) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right), \tag{2.3.8}$$

where the denominator is a scaling factor used to control the scale of the attention score, and d_k is the dimension of the query and key vectors for get a more stable gradient.

This probability determines how important each word is expressed at that position. Obviously, a word has the highest probability of attention to itself, which is the meaning of "self" in the self-attention mechanism, but it is can also pay some attention to other words related to the current word.

The fourth step is to aggregate the value vectors for each word embedding vector in the input sentence corresponding to the attention probabilities.

After getting the vector output by self-attention, it can be sent to a feed-forward neural network for classification. The intuition for the self-attention mechanism is to keep intact the values of the word(s) we want to focus on, and drop out irrelevant words (by multiplying them by tiny numbers like 0.001, for example).

Transformer: In order to learn hierarchical abstract representations through the attention mechanism, the author designed the transformer architecture, shown in Figure 2.8. The Transformer is constructed by multi-head self-attention and the feed-forward neural layer, and can be divided into encoder part and decoder part, but in our case, only encoder is introduced for the text classification task.

2.3.5.2. *Bidirectional Encoder Representations from Transformers (BERT)*

In 2018, Devlin et al. proposed the Bidirectional Encoder Representations from Transformers (BERT) model [**16**] based on Vaswani et al.'s Transformer [**67**]. On the model architecture, the Transformer is applied with two improvements. First, when encoding word embedding, it superimposes a position embedding on each word embedding; second, it stacks the transformers to form a multi-layer Multi-header self-attention mechanism. The overall flowchart of BERT is shown in Figure 2.9.

BERT's biggest contribution includes two language models for pre-training, word embedding obtained after pre-training, and a language representation model that can dynamically distinguish polysemes. The pre-trained BERT, based on 800M words from BooksCorpus and 2,500M words from English Wikipedia, can be deemed as a general NLU (Natural Language Understanding) model to support different NLP downstream tasks. It provides the dynamic word representation, which can differentiate the meaning of a polysemous word according to the context.

It also introduces two unsupervised task to improve the pre-training:

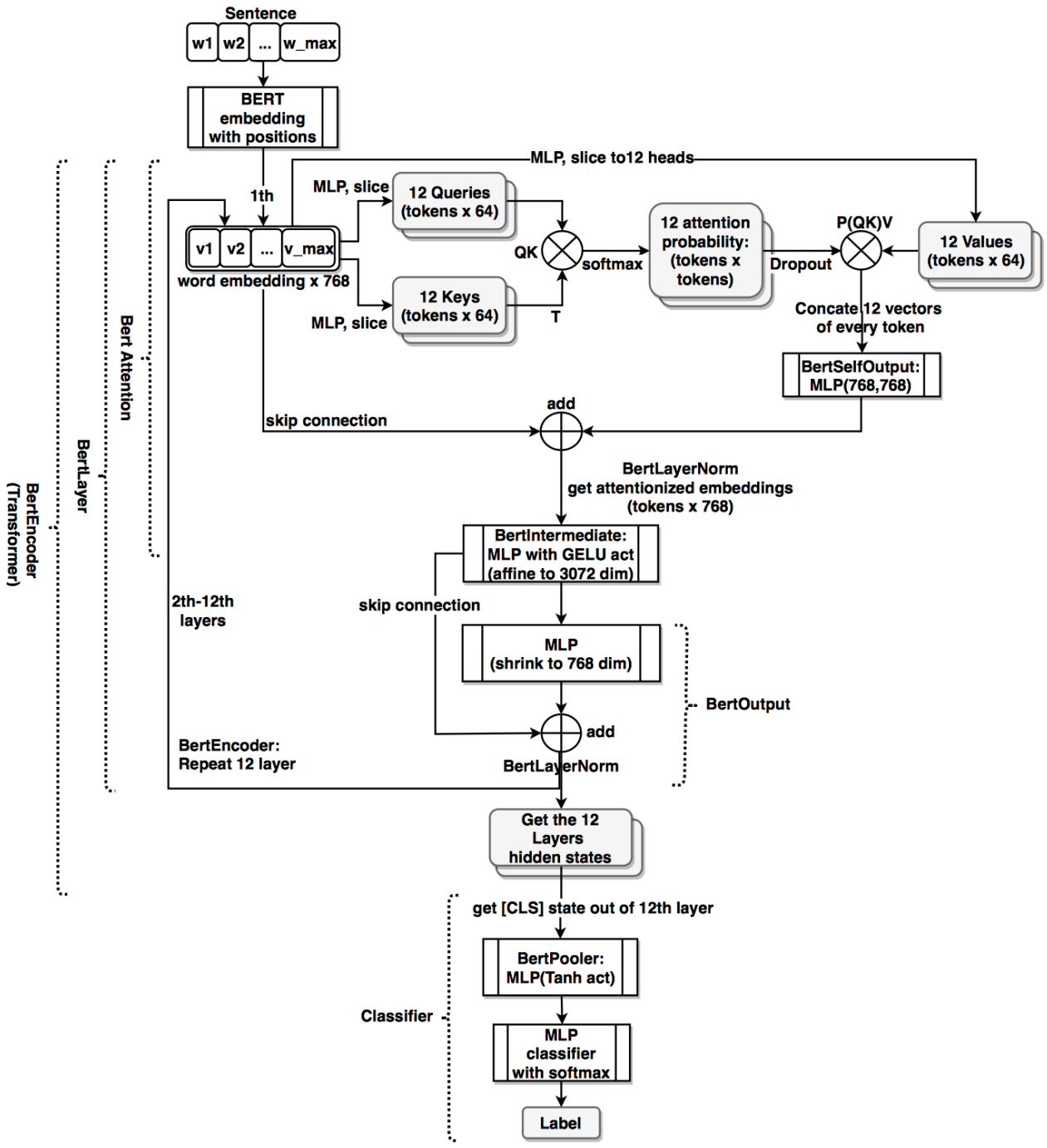


Figure 2.9. Flowchart of BERT. The number of 768 is an example of the dimension of word embedding vector, it is divided into 12 (head number) 64-dimensional word embedding vector after linear transformer.

— **Masked Language Model.** Essentially, the masked language model is a variant of the CBOW language model mentioned earlier (Figure 2.2 in Section 2.3.1.1). Rather than predicting the current word in CBOW, the Masked Language Model of BERT uses another strategy: Randomly selects 15% tokens from the corpus as training

targets, in which only 80% of them will be replaced with the [MASK] token, 10% will be replaced with a random word, and 10% of them keep unchanged.

— **Next Sentence Prediction.** BERT uses sentence classification as a pre-training task to determine if a sentence is the correct next sentence or another randomly picked sentence. This pre-training task can obtain the representation of sentence relations.

A typical input to BERT is a pair of sentences as follows [**16**]:

Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

Label = IsNext

where [CLS] is the BOS (begin of sentence) for a pair of sentences; [SEP] is used as a separator of two sentences or as EOS (end of sentence); [MASK] is used to mask out the predicted words in the masked language model. The final state of [CLS] embedding after encoding is used as the aggregated sequence representation for classification tasks. In ordinary text classification tasks, we only need to classify every individual document. For example, the movie reviews (SST-2 dataset) are usually as short as one or two sentences, so we treat every document as a sentence. Below is an example of the sentence classification:

A typical input to BERT for text classification task is a text as follows:

Input = [CLS] a very well - made , funny and entertaining picture . [SEP]

Label = Positive

The final representation of [CLS] after a number of stacked transformers is used to generate a representation of a review. This representation is used in the MLP to predict the label “Positive".

BERT model yielded the state-of-the-art performance in 11 natural language processing tasks in 2018, demonstrating its effectiveness in representation learning. However, as most of the other attention-based deep neural networks, BERT mainly focuses on local consecutive word sequences, which provides local context information. It generates a contextualized representation for a word with only its context. Thus it can be difficult for BERT to account for the global information of a language. For example, various semantic relations between words in a language may not be fully captured by BERT, such as the correlations between the words mentioned in similar contexts, limiting the representative capacity of the method.

2.3.6. Graph Neural Networks (GNNs)

GNNs work on a graph of nodes rather than a sequence of words. The goal is to create a representation for nodes that take into account their connections with neighbors.

GNN has been proposed to address one limitation of CNN: its convolution operation is a weighted accumulation for Euclidean structure data (such as image), which is not applicable to the non-Euclidean structure data. Yet there are complex graph structures in the real world such as social networks, product-shop-person relationships, molecular structures, and

so on. In 2005, Gori et al. [**58**] proposed Graph Neural Networks (GNN) to address it. In 2013, Bruna et al. [**9**] developed a variant of GNN based on spectral graph theory, and since then the GNN model has developed rapidly.

There were attempts applying GNN to NLP [**11, 5**] to capture the general knowledge about the words in a language, especially for text classification [**22, 15, 36, 54, 77**].

GNN for text classification task can be divided into Graph Convolutional Networks [**36**], Graph Attention Networks [**68**], Gated Graph Neural Networks [**43**] and other types. In the general GNN framework for document classification, documents are nodes in the graph, and the relationships between these documents are edges. The GNN model first establishes an adjacency matrix according to the graph, and uses convolution operations to aggregate all neighbours into the representation of each node for classification.

We describe two relevant architectures for text classification: GCN and Text GCN.

2.3.6.1. *Graph Convolutional Networks (GCNs)*

GCN [**36**] is one of the approaches of GNN based on spectral graph theory. It first builds a symmetric adjacency matrix based on a given relationship graph with self-loop (such as a citation relationship), followed by aggregation algorithm (usually multiplying neighbors by the weight of the edges and accumulating). At the time of the convolution, the representation of each node is fused according to the neighbor relationship in the graph. A typical GCN [**36**] is a two-layer neural network, in which the final representation of each node can convolve the neighbor nodes and the neighbor nodes of the neighbors. The overall model, a multi-layer GCN for semi-supervised learning, is schematically depicted in Figure 2.10.

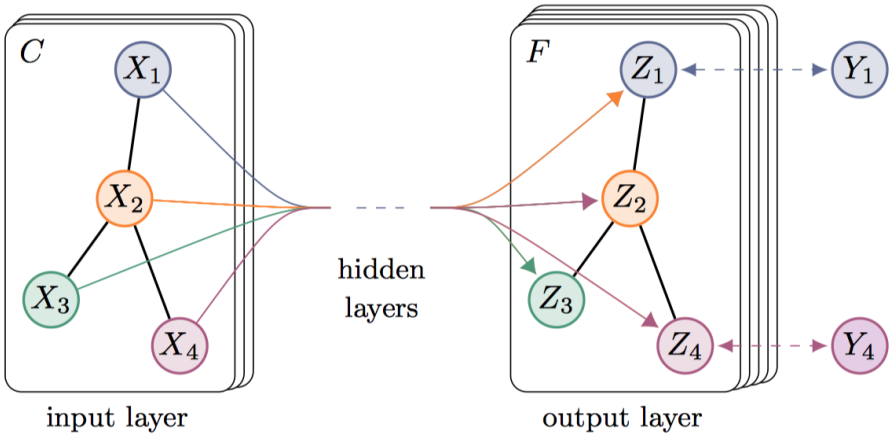


Figure 2.10. Schematic of Graph Convolutional Networks for text classification. A multi-layer Graph Convolutional Network (GCN) for semisupervised learning with C input channels and F feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by Y_i.

A general GCN [**36**] is a multi-layer neural network that convolves directly on a graph and induces embedding vectors of nodes based on properties of their neighborhoods. Formally, consider a graph $G = (V, E)$, where V (with $|V| = N$) and E are sets of nodes and edges, respectively. In GCN, the graph is a self-loop graph, where every node is assumed to be connected to itself, i.e., $(v, v) \in E$ for any v.

Usually, the adjacency matrix A and its degree matrix D is used to represent graph G, where diagonal elements of A are set to 1 because of self-loops, and D is a diagonal matrix, in which $D_{ii} = \sum_j A_{ij}$. For one convolutional layer, the formula is

$$H = \tilde{A}XW, \tag{2.3.9}$$

where $X \in \mathbb{R}^{n \times m}$ is the input matrix with n nodes and m dimensions of the feature, $W \in \mathbb{R}^{m \times h}$ is a weight matrix, and $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the normalized symmetric adjacency matrix. The normalization operation for A is to avoid numerical instabilities and exploding/vanishing gradients when used in a deep neural network model [**36**].

One layer of GCN can only capture information about direct neighbors. The study of [**36**] and [**42**] shows that a two-layer GCN performs better than a one-layer GCN, while more layers provides insignificant improvement. Then a complete two-layer GCN model is as follows,

$$Z = \text{softmax}(\tilde{A}\ \text{ReLU}(\tilde{A}XW_h)W_c), \tag{2.3.10}$$

where ReLU is an activation function, W_h is the hidden layer weight and W_c is the output layer weight.

The main problem of GCN is that it can only use the transductive learning mode, that is, one can only obtain a representation for a node (document) if it is included in the graph at training time. GCN cannot deal with unseen documents. Several methods have been proposed to address this problem. Hamilton et al. proposed GraphSAGE [**21**] to learn an embedding function with good generalization ability for unseen nodes, via feature sampling of node neighborhoods. Rather than node neighborhoods, Chen et al. proposed FastGCN via important nodes sampling and revisiting the graph convolution as integral transforms of embedding functions [**12**].

2.3.6.2. *Text Graph Convolutional Network (Text GCN)*

The traditional GCN model [**15, 54, 77**] usually studies the graph of a given node relationship [**36, 68**], such as citation network and knowledge graph. More recently, Yao et al. proposed the Text GCN [**76**], a special case of GCN, for a purpose of classify the ordinary documents without a given relationship. This produced good results.

Text GCN builds a large heterogeneous graph in which words in vocabulary and all documents correspond to nodes. The number of nodes in their graph $|P|$ is the corpus size

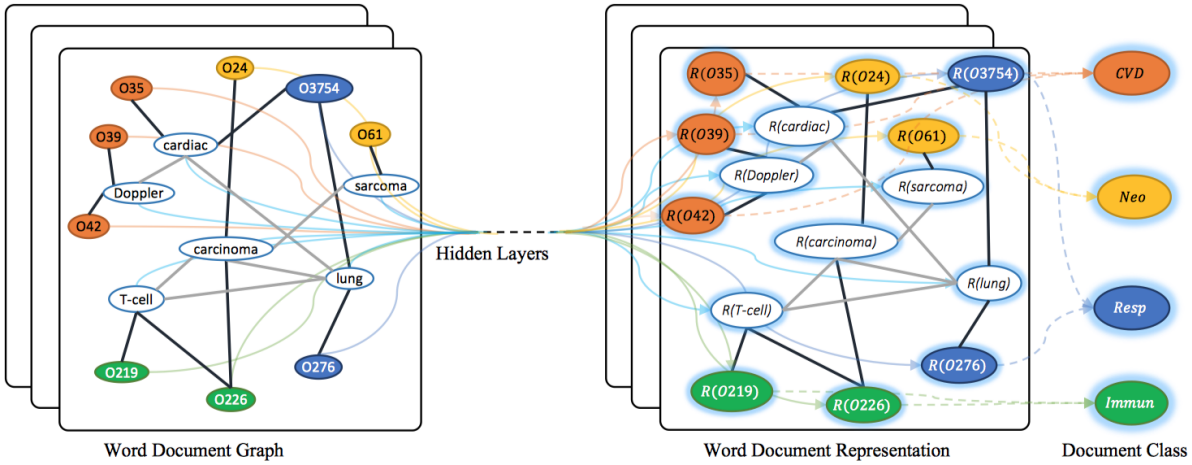


Figure 2.11. Schematic of Text GCN. In the left, nodes begin with "O" are document nodes for classification, others are word nodes. Black bold edges are document-word edges and gray thin edges are word-word edges. In the right, the nodes of R(x) means the representation after the convolution layer. "CVD, Neo, Resp, immun" are 4 categories with different colors.

(contains train set, validation set, and test set) plus the vocabulary size. The heterogeneous graph of the Text GCN model is schematically depicted in Figure 2.11.

The edges between words are determined by point-wise mutual information (PMI) across the entire corpus, and those between a word and a document by term frequency-inverse document frequency (TF-IDF). To sum up, Text GCN use the following formulas to construct a heterogeneous graph,

$$A_{ij} = \begin{cases} \text{PMI}(i,j) & i,j \text{ are words, PMI}(i,j) > 0 \\ \text{TF-IDF}_{ij} & i \text{ is document, } j \text{ is word} \\ 1 & i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.3.11}$$

where i and j represent the nodes in the graph.

After getting the adjacency matrix A, the Equation $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is used for get the normalized symmetric adjacency matrix $\tilde{A}$. Then Text GCN uses the same Formula 2.3.10 to train the model. However, the feature matrix X is an identity matrix, which means every word or document is represented as a one-hot vector as the input to Text GCN. Therefore a two-layers Text GCN model is as follows,

$$Z = \text{Softmax}\left(\tilde{A}\ \text{ReLU}(\tilde{A}IW_h)W_c\right) \tag{2.3.12}$$

where ReLU is an activation function, $ReLU(x) = \max(0,x)$. W_h is the hidden layer weight and W_c is the output layer weight corresponding to different labels.

GCN and Text GCN are good at convolving the global information in the graph into a sentence, but they do not take into account local information such as the order between words, which is important to understand the document. Therefore, it is natural to combine GCN with a model capturing local information such as BERT.

2.3.7. Existing Combinations of GCN and BERT and Outline of Our Approach

Some recent studies have combined GCN with BERT. Shang et al. [**60**] applied a combination to the medication recommendation task, which predict a medical code given the electronic health records (EHR), i.e., a sequence of historical medical codes, of a patient. They first embed the medical codes from a medical ontology using Graph Attention Networks (GAT), then feed the embedding sequence of the medical code in an EHR into BERT for code prediction. In this study, there is no order in the code sequence, while in text classification, the order of words is important.

Jong et al. [**27**] proposed another combination to the paper citation recommendation task using paper citation graphs. This model simply concatenates the output of GCN and the output of BERT for downstream predictive tasks. No interaction is allowed between GCN and BERT in their operations.

We believe that interactions between the local and global information are important and can benefit the downstream prediction tasks. In fact, through layers of interactions, one could allow the information captured in GCN to be applied to the input text, and the representation of the input text to be spread over GCN. This will produce the effect we illustrated in Chapter 1, on the movie review example (*a way few current films do* vs. *innovation*), and the offensive language example (*perverted religion* vs. *forcing imam*). This is the approach we propose in our study, which will be described in the next chapter.

Chapter 3

PROPOSED METHOD

In this chapter, we describe our proposed method based on our motivation. We first build a vocabulary graph that can represent global language information (in particular, the relations between words). Then we introduce the vocabulary graph convolutional networks based on the vocabulary graph. Finally, we propose our method to combine the Vocabulary GCN with BERT.

As aforementioned in Chapter 1 and Sub-section 2.3.5.2, the BERT (Bidirectional Encoder Representations from Transformers) [**16**], which leverages a multi-layer multi-head self-attention (called transformer) together with a positional word embedding, is one of the most successful deep neural network model for text classification in the past years. However, as most of the other attention-based deep neural networks, BERT mainly focuses on local consecutive word sequences, which provides local context information. That is, a word is placed in its context, and this generates a contextualized representation. However, it may be difficult for BERT to account for the global information of a language.

On the other hand, Graph Convolutional Networks (GCN) and its variants are good at convolving the global information in the graph, which create representations for nodes by aggregating information from their neighbors. However, they do not take into account local information such as the order between words in a sentence. When word order and other local information are important in a task, GCN may be insufficient. Therefore, it is natural to combine GCN with a model capturing local information such as BERT.

In the following sections, we will describe how our vocabulary graph is constructed and encoded. Then we will describe its integration into BERT for text classification.

3.1. Vocabulary GCN (VGCN)

The traditional GCN models cannot be directly integrated into BERT. On the one hand, they directly perform convolution operations on the graph containing document nodes and relations between documents, and propagate the classification information to the neighbor documents (coarse granularity) to enhance the classification performance. In our case, we

want to perform convolution operations on words using relations between words at a finer granularity. On the other hand, the traditional GCN is transductive-learning mode [**2**], its training process must include the test data in the graph, which is unrealistic in practical applications. In addition, when the graph is integrated with BERT, we have to use the mini-batch training mode of BERT, which becomes difficult for a document graph. At present, no GCN model can meet these two requirements at the same time: operate on fine-grained word relations and be inductive. Therefore, we propose to use a Vocabulary GCN model to replace the document-level graph in GCN. The vocabulary graph contains words in a language. Any document can be mapped to such a graph using the words it contains. Therefore, the trained vocabulary graph can be applied to any document, without knowing the test document in advance.

3.1.1. Notion of Graph Convolutional Network (GCN)

The notions of graph convolutional network (GCN) have been described earlier. To ease the reading of this chapter, we briefly review them. A graph is defined as $G = (V, E)$, where $V(|V| = N)$ is a set of nodes and E is a set of edges. GCN aims to learn node embeddings in the graph via information propagation in neighborhoods. To leverage the connectivity structure of the graph, assuming $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix of G, each layer of GCNs can be described as:

$$L^{(i+1)} = \sigma(\hat{A}L^{(i)}W_i) \tag{3.1.1}$$

where $\hat{A} = D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}$ is the normalized symmetric adjacency matrix with self-loop for graph G, D is the degree matrix of A, I is an identity matrix, $L^{(i)}$ is the ith layer feature matrix for all nodes, W_i is the weight matrix for ith layer, and σ is an activation function. When applied to learn text representations, documents of the whole corpus are represented as nodes in a large graph, embeddings of them can be learned by stacking multi-layer GCN to capture multi-hop neighborhood information with such propagation process.

As with most other graph-based algorithms, the adjacency matrix is required to encode the pairwise relationships for both training and test data to induce their representations. However, in many applications, test data should not available during the training process. The transductive learning approach cannot cope with the problem. This requires the method to be inductive, i.e. to infer the representations of the unseen texts according to only the information of the texts available in training.

3.1.2. Building Vocabulary graph (VGraph)

To infer out-of-graph samples, we propose the vocabulary GCN to make it feasible for inductive inference. The idea is depicted in Figure 3.1 and Figure 3.2, in which a document graph is translated into a vocabulary graph. In this subsection, we start from the following

two key problems: 1) How to build a document-independent graph without losing the original document correlation information? 2) How to effectively incorporate the word co-occurrence and document adjacency graph to learn document representations?

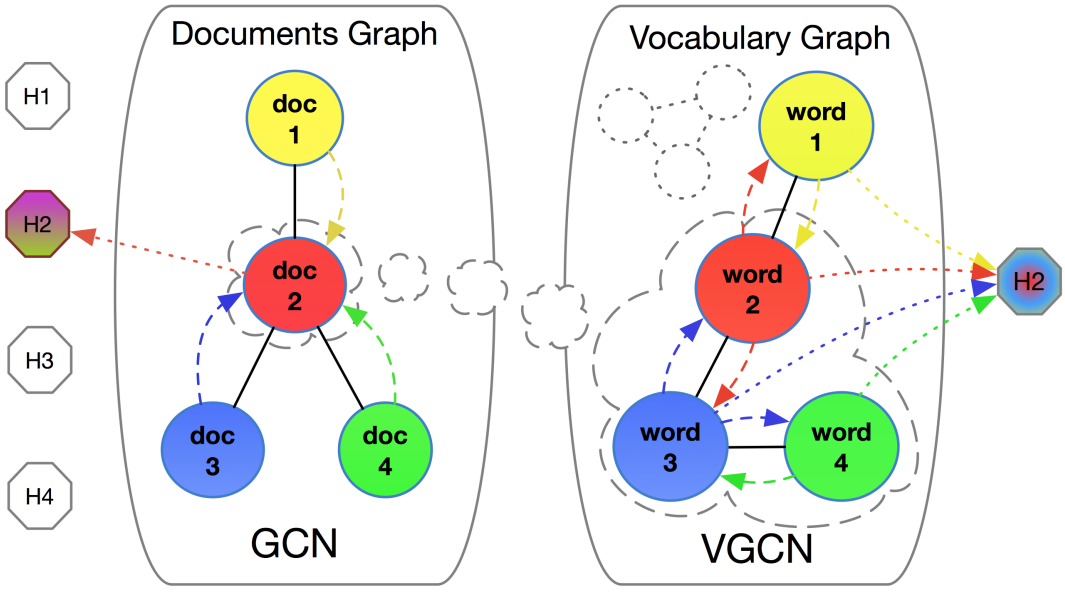


Figure 3.1. Illustration of GCN and VGCN for one document. In GCN, the representation $H2$ of document 2 is aggregated by all other documents (doc1, doc3, doc4) in the document-level graph. In VGCN, the representation $H2$ of document 2 is aggregated by all words of document 2 (word2, word3, word4) and other words connected (word1) in the vocabulary graph.

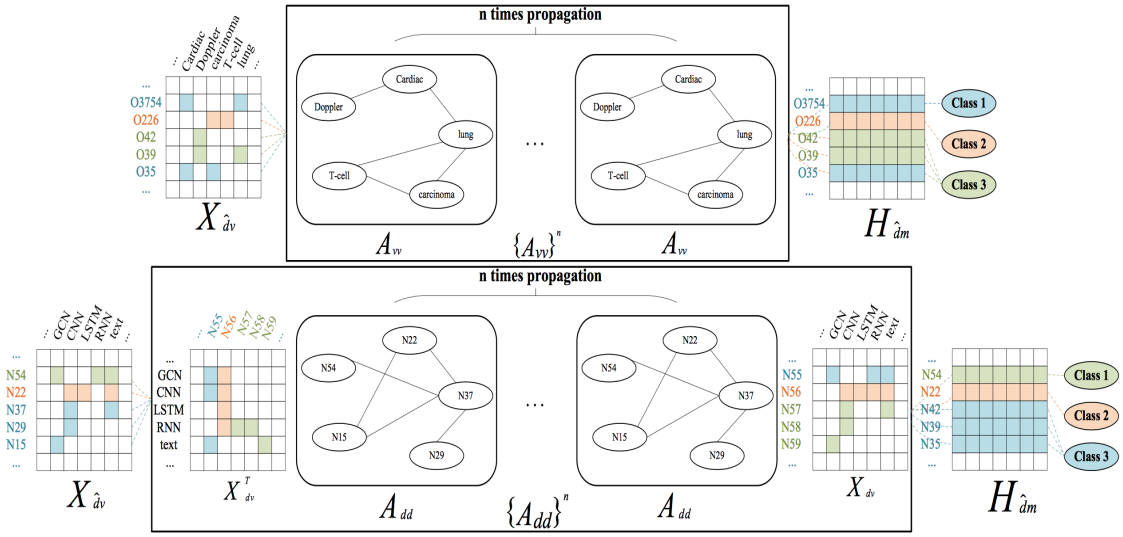


Figure 3.2. Schematic of VGCN model, in which the top row shows the word convolutional via n times propagation on the VGraph A_{vv} without the document correlation graph, the bottom row shows the word convolutional via n times propagation on the document correlation graph A_{dd}.

Building Word-level Graph (Vocabulary Graph): Given a corpus $D(d = |D|)$ with vocabulary size $v = |V|$, we build a flexible word-level vocabulary graph (VGraph), whose nodes are all words in V. We consider two cases: the documents are independent, and the documents are connected.

1. Suppose that documents are independent, given the document-word features X_{dv} of the corpus D, we can directly represent a word by observing whether it appears (or possibly with the term frequency) in all documents, i.e., $\vec{w_i} = [x_{i1}, x_{i2}, \cdots, x_{id}]$. A correlation between two words can be calculated by the inner product between two word features vectors i.e., $c_{ij} = \vec{w_i}\vec{w_j}^T$. Note that the i-th word representation $\vec{w_i}$ is i-th column of X_{dv}. Thus, correlations between two arbitrary words can be defined as:

$$A_{vv} = X_{dv}^{\mathrm{T}} X_{dv} \in \mathbb{R}^{v \times v} \tag{3.1.2}$$

which produces a matrix of dimensions $v \times v$. This is also a way to express the co-occurrence of words in the documents. With proper normalization, the above between-word correlation matrix A_{vv} can be transformed into an adjacency matrix to build a graph. For example, a Point-wise Mutual Information (PMI, or normalized point-wise mutual information (NPMI)) matrix is one of the typical word correlation matrices to use [**76**].

2. Now, suppose that documents are not independent and are connected with some relation, for example, citation relation. When such inter-document connections are present, we can define a more general word correlation matrix. Technically, the i-th word feature with n-order inter-document relationships is defined as follows:

$$\vec{w_i}^{(n)} = [x_{i1}, x_{i2}, \cdots, x_{id}] \underbrace{A_{dd} \cdots A_{dd}}_{\text{n times}} = \vec{w_i}\{A_{dd}\}^n \tag{3.1.3}$$

n determines how many hops of citation relationships the word representation should capture. A bigger n of $\vec{w_i}^{(n)}$ means a longer hop citation will be leveraged. Based on the order of inter-document relationship used for word representations, a general definition of word correlations is as follows:

$$A_{vv}^{(m,n)} = X_{dv}^T \{A_{dd}\}^m \{A_{dd}{}^T\}^n X_{dv} \tag{3.1.4}$$

where $A_{vv}^{(m,n)}$ denotes the correlations between word representations utilizing m-order and n-order inter-document relationships.

According to previous methods of GCN [**36**], A_{dd} will be transformed into a symmetric Laplacian matrix. Thus we can rewrite the formula in Equation 3.1.4 as follows:

$$A_{vv}^{(k)} = X_{dv}^T \{A_{dd}\}^k X_{dv} \tag{3.1.5}$$

where $k = m + n$. In the document-independent VGraph, we effectively unify both word co-occurrence ($X_{dv}^T X_{dv}$) and the inter-document relation information (A_{dd}), to make them complement each other to improve the document representation learning. In this convolution

operation, the first word representation X_{dv}^T will aggregate the relationship of the citation relationship A_{dd}, and then multiplied by the second X_{dv} to obtain the final correlation, which means that the word co-occurrence matrix is aggregated by the citation relationship.

Vocabulary Graph Convolution: We propose the vocabulary graph convolutional network (VGCN) to embed VGraph for learning document representations.

First, we apply the GCN Equation 3.1.1 on the adjacency matrix A_{vv} to learn the representations of words:

$$H_{vm} = \sigma(\{A_{vv}\}^n X_v W_0) \tag{3.1.6}$$

where $H_{vm} \in \mathbb{R}^{\hat{v} \times m}$ is the word embedding matrix, σ is an activation function, and $W_0 \in \mathbb{R}^{v \times m}$ is the weight matrix. X_v is the feature matrix for all words, it can be X_{dv}^T in 3.1.2. Since we have calculated A_{vv} using X_{dv}^T in 3.1.2, here we simply set word feature matrix $X_v = I \in \mathbb{R}^{v \times v}$ as an identity matrix which means every word of documents is represented as a one-hot vector to represent itself.

Then we multiply the document-word feature vector $H_{\hat{d}m}$ with the word representation vector H_{vm}, and finally transformed into the document representations. Then our VGCN equation is as follows:

$$H_{\hat{d}m} = X_{\hat{d}v} H_{vm} = X_{\hat{d}v} \sigma(\{A_{vv}\}^n X_v W_0) \tag{3.1.7}$$

where $H_{\hat{d}m} \in \mathbb{R}^{\hat{d} \times m}$ is the document embedding matrix, and $X_{\hat{d}v} \in \mathbb{R}^{\hat{d} \times v}$ is the feature matrix of input documents. It is noticed that $X_{\hat{d}v}$ is different from X_{dv}: X_{dv} is fixed as the document-word feature matrix of training documents, while $X_{\hat{d}v}$ can be the feature matrix of training or test data, which enables our model to infer representations for seen or unseen documents.

Similar to [**73**], we apply a nonlinear activation function after n times message propagation to simplify the model structure, resulting in a linear structure with the n-th power adjacency matrix as $\{A_{vv}\}^n$ and a single weight matrix W. After learning the word node embeddings in the VGraph, the document-word feature matrix of the input data $X_{\hat{d}v}$ is utilized to generate final document representations $H_{\hat{d}m}$.

As for the scenario with the provided document correlation graph, the propagation rule for the adjacency matrix A_{vv} is different:

$$H_{\hat{d}m} = X_{\hat{d}v} \sigma((X_{dv}^{\mathrm{T}} \{A_{dd}\}^n X_{dv}) X_v W_0) \tag{3.1.8}$$

where $A_{dd} \in \mathbb{R}^{d \times d}$ is the given document correlation graph, and $X_v = I \in \mathbb{R}^{v \times v}$ is also an identity matrix. Only the n-th power of A_{dd} is utilized to perform the message propagation while X_{dv} does not participate in the propagation. This is done to fully leverage the pairwise information between documents in the predefined graph. Overall, our model is flexible to applicable for both without and with the document-level graph scenarios.

For the classification task, we construct a word graph using the words in the documents. The words are connected according to heir NPMI determined from the document collection. hen we learn a document representation by VGCN, which is then fed into a simple MLP layer, followed by the softmax classifier to predict target labels. The loss function can be defined as:

$$\begin{aligned} \hat{Y}_{\hat{d}f} &= softmax(ReLu(H_{\hat{d}m})W_1 + b_1) \\ \mathcal{L} &= -\sum_{i \in \hat{d}} \sum_{j \in f} Y_{i,j} ln \hat{Y}_{i,j} \end{aligned} \tag{3.1.9}$$

where $W_1 \in \mathbb{R}^{m \times f}$ is the weight matrix, b_1 is the bias, Y is the label indicator matrix, f is the number of classes.

3.2. Intergrating VGCN into BERT

In the previous section, we proposed a more versatile VGCN approach, which is suitable for the situation with a given document graph (such as citation network), as well as general text classification tasks. In this section, we only discuss the combination of VGCN and BERT in the absence of a given document graph (such as citation network), i.e., in the case of general text classification (such as offensive language detection). As a future work, we can continue to study VGCN-BERT in the case of a given document graph.

3.2.1. Simplifying VGCN and VGraph

Simplifying VGCN: We need to adapt VGCN to the situation of mini-batch training before integrating VGCN into BERT, because BERT is a heavy model (110 million parameters), it is necessary to perform the mini-batch training. Besides, in the following experiments of VGCN, we found that the performance (Table A.3) is the highest when only one order of vocabulary graph (i.e. A^1_{vv}) is convolved, and since $X_v = I \in \mathbb{R}^{v \times v}$, we put Equation 3.1.7 into Equation 3.1.9 into a single equation as follows:

$$\mathbf{VGCN}_{\text{mini-batch}} = softmax(ReLu(X_{mv}A_{vv}W_{vh})W_{hc} + b_c), \tag{3.2.1}$$

where m is the mini-batch size, v is the vocabulary size, h is the hidden layer size, and c is the number of classes. Every row of X_{mv} is a vector containing document features, which can be a bag-of-words vector or TF-IDF. The above equation refers to a 1-order VGraph convolution layer, which captures the part of the graph relevant to the input (through $X_{mv}A_{vv}$), i.e. encoding the word mentioned in the input with their related words in vocabulary graph.

When combined with BERT, VGCN acts as a component that provides the graph representation, and its input is the word embedding of BERT. We call this component VGCN graph embedding, and its corresponding equation is as follows:

$$\mathbf{VGCN}_{\text{graph-embedding}} = ReLu(X_{mev}A_{vv}W_{vh})W_{hg} + b_g, \tag{3.2.2}$$

where W_{hg} and b_g, which were originally used for classification, become the output of size g of graph embedding (hyperparameter) whose dimension is the same as the word embedding; m is the size of the mini-batch, and e is the dimension of word embedding, and v is the vocabulary size.

Note that the major difference between Equation 3.2.2 and Equation 3.2.1 is that, the dimension shape of the word embedding from BERT is (m,e,v), but the original X of Equation 3.2.1 is just a matrix (m,v) from BoW or TF-IDF. Thus, to adapt Equation 3.2.1 to the 3-dimensional mini-batch word embedding, we just keep m and e, and apply every slice of $\vec{x}$ to Equation 3.2.1, resulting in the generated graph embedding with the dimension of $m \times e \times g$, where m is the size of the mini-batch, e the dimension of word embedding, and g the size of graph embedding.

Simplifying VGraph: Our goal is to obtain application-dependent (such as offensive language detection) global language information to complement the original local information. The global language information can take multiple forms (e.g. word co-occurrences, Wordnet). The statistical word co-occurrences reflect rich similarity information between words. Yao et al. [**76**] used word co-occurrences for Text GCN model and got good results. Therefore, we directly construct a 1-order VGraph using the statistical word co-occurrences, instead of using Equation 3.1.2 to calculate the word-word relations. The advantage of doing this is that we can use an adequate window size to control the granularity of the co-occurrence information we want to obtain, which cannot be done using Equation 3.1.2. Therefore, we adopt the same approach.

We use the normalized point-wise mutual information (NPMI) to calculate the word co-occurrences information, then construct the VGraph for VGCN-BERT. [**8**] recommends using NPMI instead of PMI: the NPMI is easier to be combined with pre-trained word embedding of BERT, and can reduce the problem of gradient explosion. The equation of NPMI is shown as follows:

$$\text{NPMI}(i,j) = -\frac{1}{\log p(i,j)} \log \frac{p(i,j)}{p(i)p(j)} \tag{3.2.3}$$

where i and j are words, $p(i,j) = \frac{\#W(i,j)}{\#W}$, $p(i) = \frac{\#W(i)}{\#W}$, $\#W(*)$ is the number of sliding windows containing a word or a pair of words, and $\#W$ is the total number of sliding windows. The range of NPMI is [-1,1]. A positive NPMI value implies a high semantic correlation of words, while a negative NPMI value indicates little or no semantic correlation.

By setting a threshold, NPMI VGraph can also reduce the density of the adjacency matrix of VGraph, and speed up the operation speed of VGCN. For example, when the corpus is large, in order to maintain the sparsity of the matrix of A, we can set a threshold of NPMI

to take a part of the [-1,1] interval. In our approach, we create an edge between two words if their NPMI is larger than a threshold. Our experiments show that the performance is best when the threshold is between 0.0 and 0.3.

3.2.2. Vanilla VGCN-BERT

Jeong et al. has been proposed a simple combination model of VGCN and BERT in [**27**], which we call Vanilla-VGCN-BERT. The original version is applied to the medication recommendation task, we adapt this model to the text classification task.

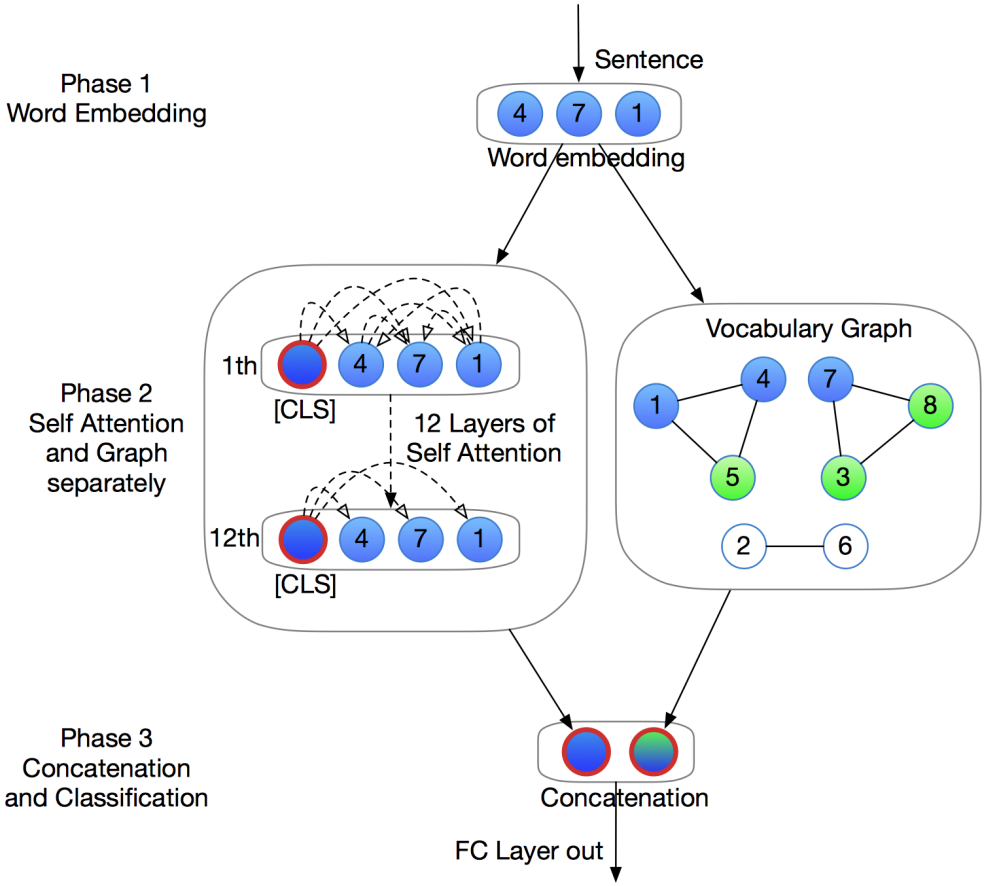


Figure 3.3. Illustration of Vanila-VGCN-BERT. The same word embedding sequence is input to two independent models (BERT and VGCN), and then the two output representations are simply concatenated for classification.

In this method, the pre-trained-BERT's word embedding of the document is fed into VGCN and BERT independently, then the graph representation and the BERT's document representation are obtained respectively, and finally the new concatenated representation will be fed to the fully connected layer for the classification. The overall Vanilla-VGCN-BERT model is schematically illustrated in Figure 3.3. The equation for training Vanilla-VGCN-BERT is as following:

$$\begin{aligned} \mathrm{H}_{\text{concatenated}} &= \mathrm{VGCN}_{\text{graph-embedding}}(X_{mev}) \bigoplus \mathrm{BERT}(X_{mes} + P_{mes}), \\ \textbf{Vanilla-VGCN-BERT} &= softmax(\mathrm{H}_{\text{concatenated}} W_c + b_c), \end{aligned} \tag{3.2.4}$$

where $\bigoplus$ is the vector concatenation operation, X_{mev} is the extended word embedding of BERT (padding zero vectors for the non-appeared words in a sentence to the vocabulary size), X_{mes} is the original word embedding, P_{mes} is the position embedding for the original sentence sequences, W_c and b_c is the full-connected layer for classification.

The Vanilla-VGCN-BERT model simply concatenates the output of GCN and the output of BERT for downstream fully-connected prediction layer, that is, the fully-connected layer's weights can only be learned via two final representations of the document (BERT's output and VGCN's output). However, these two representations are high-level and abstract. Therefore it cannot fully learn the correlation between local words and background words.

3.2.3. Interactive VGCN-BERT

Intuitively, The local and global information can influence mutually. For example, if two words in a sentence are not connected globally (no similarity between them), then they can possibly be connected through self-attention in BERT, which is local information. On the other hand, if these words are globally connected in the vocabulary graph, then BERT should allocate a stronger self-attention between them, and this will produce a different representation at higher level. This example shows the possible influence between local an global information. We use the process illustrated in Figure 3.4 for the integration: We feed the original sentence (local information using word embedding of BERT) together with the global language information (application-dependent co-occurrence words convolved by VGCN) to the self-attention encoder in BERT, the multi-level multi-head self-attention mechanism will fully interact with local information and global language information. Finally we use the final representation (sentence embedding) out from the last layer of BERT to classify.

In order to tightly integrate VGCN component into BERT, We first decompose BERT. When BERT is applied to text classification, a typical solution contains three parts (refer BERT framework in Figure 2.9): (1) word embedding module with the positional information of the word; (2) transformer module (refer transformer framework in Figure 2.8) using multi-layer multi-head self-attention stacking; and (3) fully connected layer using the generated sentence embedding for classification.

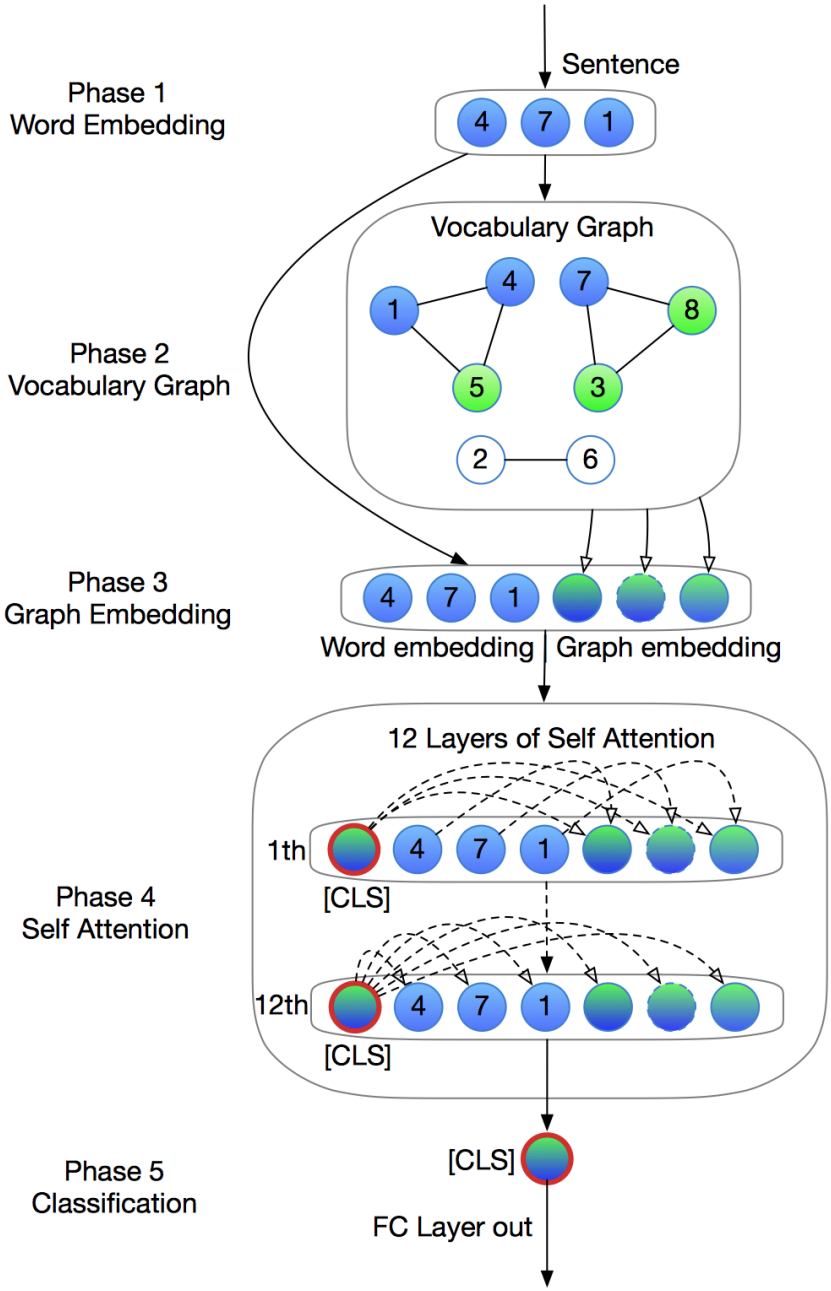


Figure 3.4. Illustration of Interactive-VGCN-BERT. Blue represents the currently entered words, green represents the associated words in the graph, and the blue-green blend represents the convolutional representation. The embeddings of input sentence (Phase 1) are combined with the vocabulary graph (Phase 2) to produce a graph embedding, which is concatenated to the input sentence (Phase 3). In this demonstration, phase 2 produces three (hyperparameter) graph embeddings which has the same shape as word embedding. Note that from the vocabulary graph, only the part relevant to the input is extracted and embedded, in this case (3,5,8) will be convoluted in but (2,6) will not. In Phase 4, several layers of self-attention are applied to the concatenated representation, allowing interactions between word embeddings and graph embedding. The final embedding at the last layer is fed in a fully connected layer (Phase 5) for classification.

Recall that self-attention operates with a query Q against a pair of key K and value V. The attention score is calculated as follows:

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3.2.5}$$

Using these attention scores, the method can yield a weighted vector representation encoding the contextual information for all mentioned words.

Instead of using only word embeddings of the input sentence in BERT, we feed both the vocabulary graph embeddings obtained by the VGCN-graph-embedding component (Equation 3.2.2) and the sequence of word embeddings to BERT transformer. Thus our method considers the order of the words in the sentence and the background information captured by VGCN simultaneously. The overall Interactive-VGCN-BERT model is schematically illustrated in Figure 3.4. Through the attention score calculated by Equation 3.2.5, local embedding and global embedding are fully integrated after layer-by-layer interaction in 12-layer and 12-heads self-attention encoder.

The equation for training Interactive-VGCN-BERT is as following:

$$\begin{aligned} G_{meg} &= ReLu(X_{mev}A_{vv}W_{vh})W_{hg} + b_g, \\ H_{me} &= \text{Transformer}((X_{mes} + P_{mes}) \bigoplus G_{meg}), \\ \textbf{Interactive-VGCN-BERT} &= softmax(H_{me}W_{ec} + b_c), \end{aligned} \tag{3.2.6}$$

where G_{meg} is the VGCN graph embedding with size g (it is the same as Equation 3.2.2), and $\bigoplus$ is the vector concatenation operation, X_{mev} is the extended word embedding of BERT (padding zero vectors for the non-appeared words in a sentence to the vocabulary size), X_{mes} is the original word embedding, P_{mes} is the position embedding for the original sentence sequences, W_{ec} and b_c are weights and bias used in the full-connected layer for classification.

Compared to Vanilla-VGCN-BER, this method has a better capability to integrate local and global information through the self-attention mechanism. We expect it will produce better classification results than Vanilla-VGCN-BERT. This will be shown in our experiments described in the next chapter.

Chapter 4

EXPERIMENTAL RESULTS

In this chapter, we will show experimental results with our model. We focus on the use of our VGCN-BERT model on hate language detection. We will also test on several other text classification datasets to show that the approach can be applied to any text classification task.

In our experiments on hate language detection, the documents are independent. So we cannot use the full capacity of our VGCN, which can also work on documents that are connected. So, we will perform experiments on the full VGCN on some other document collections in which documents are connected. As the collections are not about hate language detection, we will put these experiments in an appendix.

We will evaluate our VGCN-BERT model and compare it with several baselines. The main questions we aim to study in this experiment are:

— Can VGCN-BERT models using both global and local information do better than BERT and VGCN separately, and other baselines?

— Is it useful to allow interactions between local and global information?

4.1. Baselines

In addition to the original BERT model, we also use several other neural network models as baselines, as well as 3 typical traditional machine learning models (i.e. KNN, Naive Bayes, SVM).

— **MLP**: Multilayer perceptron with 2 hidden layers, and bag-of-words model with TF weighting.

— **Bi-LSTM** [**20**]: The BERT's pre-trained word embeddings are used as input to the Bi-LSTM model.

— **Text GCN**: The original Text GCN model uses the same input feature as MLP model, and we use the same training parameters as in [**76**].

— **VGCN**: This model only uses VGCN, corresponding to Equation 3.2.2, but the output dimension becomes the class size. BERT's pre-trained word embeddings are used

Table 4.1. Summary statistics of five datasets for evaluation of VGCN-BERT

Dataset	#Docs	#Test	#Classes	#classes ratio	Average Length
ArangoHate	7,006	700	2	1.4:1	13.3
FountaHate	99,996	9999	4	10.9:5.5:2.8:1	15.7
SST-2	9,613	1,821	2	1:1	19.3
CoLA	9,594	1,043	2	2.4:1	7.7

as input. The output of VGCN is relayed to a fully connected layer with Softmax function to produce the classification score. This model only uses the global information from vocabulary graph.

— **BERT**: We use the small version (*Bert-base-uncased*) pre-trained BERT [**16**]. It uses 12-layer and 12-heads Transformer, and its dimension of word embedding is 768, the count of parameters is 110M.

— **Vanilla-VGCN-BERT**: To compare with Interactive-VGCN-BERT model, we implemented a vanilla combination of BERT and VGCN similar to [**27**], which produces two separate representations through BERT and GCN, and then concatenates them. ReLU and a fully connected layer are applied to the combined representation for classification. The main difference of this model with our methods is that there is no interactions between the input text and the graph.

4.2. Datasets

We test the models on two hate language detection datasets: ArangoHate and Funtta-Hate. As our model can also be used for general text classification, we also do experiments on two additional datasets: SST-2 and CoLA. Notice that all these test collections only contain short texts. We are unable to run our model on long texts due to the limitation of our computational capacity of GPU: BERT is a heavy-weight deep learning model (110 million parameters for the *Bert-base-uncased* version). For example, using the parameters as 16-size mini-batch containing the 256-length sequence to train the small version VGCN-BERT model will consume about 8-10G of GPU graphical memory. The statistics of 4 datasets is shown in Table 4.1. Below are some more details about the datasets.

— **ArangoHate** [**3**] is a resampled dataset merging the dataset from [**72**] and the dataset from [**14**]. It contains 2,920 hateful documents and 4,086 normal documents. The average length is 13.3 words. Since the dataset is not pre-divided into train, valid and test sets, we split it into three sets at the ratio of 85:5:10 after shuffle.

— **FountaHate** is a large four-label dataset for hate speech and offensive language detection [**17**]. It contains 99,996 [1] tweets with cross-validated labels and is classified into 4 labels, normal (53,851), spam (14,030), hateful (27,150) and abusive (4,965).

1. The final version provided by the author is more than the one described in the paper.

The average length is 15.7 words. Since the dataset is not pre-divided into training, validation, and test sets, we split it into three sets at the ratio of 85:5:10 after shuffle.

— **SST-2** The Stanford Sentiment Treebank is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment [**62**]. We use the public version [2] which contains 6,920 examples in the train set, 872 in the valid set, and 1,821 in the test set, for a total of 4,963 positive reviews and 4,650 negative reviews. The average length of reviews is 19.3 words.

— **CoLA** The Corpus of Linguistic Acceptability is a binary single-sentence classification task. CoLA is manually annotated for acceptability (grammaticality) [**71**]. We use the public version which contains 8,551 train data and 1,043 valid data [3], for a total of 6,744 positive and 2,850 negative cases. The average length is 7.7 words. Since we do not have the label for the test set, we split 5% from the shuffled train set as a valid set and use the original validation set as the test set.

As the label distribution in datasets CoLA (2.4 : 1), ArangoHate (1.4 : 1) and FountaHate (10.9 : 5.5 : 2.8 : 1) are uneven, we train all models using weight for each class that are reversely proportional to the size of the class.

4.3. Preprocessing and setting

We removed URL strings and @-mentions to retain the text content, then the text was lower-cased and tokenized using NLTK's *TweetTokenizer* [4]. We use BERTTokenizer function to split text, so that the vocabulary for GCN is always a subset of pre-trained BERT's vocabulary. When computing NPMI on a dataset, the whole sentence is used as the text window to build the vocabulary graph. The threshold of NPMI is set as 0.2 for all datasets to filter out non-meaningful relationships between words.

In the Interactive-VGCN-BERT model, the graph embedding output size is set as 16, and the hidden dimension of graph embedding as 128. We use the *Bert-base-uncased* version of pre-trained BERT, and set the max sequence length as 200. The model is then trained in 9 epochs with a dropout rate of 0.2. The following are other parameter settings for different datasets.

— SST-2: mini-batch = 16, learning rate = 1e-5, and L_2 loss weight decay = 0.01.

— CoLA: mini-batch = 16, learning rate = 8e-6, and L_2 loss weight decay = 0.01.

— ArangoHate: mini-batch = 16, learning rate = 1e-5, and L_2 loss weight decay = 1e-3.

— FountaHate: mini-batch = 12, learning rate = 4e-6, and L_2 loss weight decay = 2e-4.

These parameters are set based on our preliminary test. We also use the default fine-tuning learning rate and L_2 loss weight decay as in [**16**].

2. https://github.com/kodenii/BERT-SST2
3. https://github.com/nyu-mll/GLUE-baselines
4. http://www.nltk.org/api/nltk.tokenize.html

For the baseline of the original BERT model and VGCN model, the parameter settings are the same as the Interactive-VGCN-BERT. For the 2 hidden layers MLP, we use the term-frequency (TF) as inputs feature, and set the dimension of first fully connected layer as 512 and the second hidden layer dimension as 100, learning rate as 1.5e-3 and L_2 loss weight decay as 2e-5, batch size as 64, total train epoch as 100, and early stopping as 10. For the Text GCN, we use the same inputs feature as MLP, and the same training parameters as [**16**]: 200 total train epoch, 20 early stopping, 200 hidden dimension, 0.02 learning rate, and 0 for L_2 loss weight decay. The only difference is that we set the dropout rate as 0.2 in order to be consistent with other models. For Bi-LSTM, we use the word embedding from pre-trained *Bert-base-uncased* version, with the word embedding dimension 768, and we set the hidden dimension of Bi-LSTM as 100, the fully connected layer dimension as 50, learning rate as 1e-4 and L_2 loss weight decay as 0, batch size as 32, total train epoch as 20.

For the 3 classical machine learning models, the parameter settings are as follows: For SVM, we use the same feature extraction method as the MLP model as input, and use linear as the kernel. For Naive Bayes, when we use 1-gram and 2-grams together as vocabulary and feature, the performance is better. For KNN, we use PCA to reduce the feature dimension to 1000, and set K = 10.

4.4. Loss Function

We use cross-entropy as the loss function for all models, except for *FountaHate* dataset where we use the mean squared error as the loss function to utilize the annotators' voting information. More specifically, the "vote" column of *FountaHate* dataset refers to the number of the annotators who have provided the majority label, then we divide it by the number of annotators to get the confidence of ground truth, and use MSE to fit this confidence.

We use Adam as the training optimizer for all models. For cases where the label distributions are uneven (CoLA, ArangoHate and FountaHate), *comput_class_weight* function [5] from scikit-learn [**10**] is used as the weighted loss function. The weight of each class (W_c) is calculated by

$$W_c = \frac{\#samples}{\#classes \cdot \#samples\_in\_c}, \tag{4.4.1}$$

where $\#samples$ is the total number of samples in the dataset and $\#classes$ is the number of classes and $\#samples\_in\_c$ is the number of samples in the class c.

5. https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html

4.5. Evaluation Metrics

We adopt the two most widely used metrics to evaluate the performance of the classifiers: weighted average F1-score [**41, 48**].

$$\textbf{Weighted avg F1} = \sum_{i=1}^{C} F1_{c_i} * W_{c_i} \tag{4.5.1}$$

and macro F1-score,

$$\textbf{Macro F1} = \frac{1}{C} \sum_{i=1}^{C} F1_{c_i} \tag{4.5.2}$$

4.6. Experimental Result and Discussion

The main results on weighted average F1-Score and macro F1-Score on test sets are presented in Table 4.2. The main observation is that Interactive-VGCN-BERT outperforms all the baseline models, demonstrating the advantage of the combination.

For three classical machine learning models, we see that their performances lag behind the deep learning models (except for the Naive Bayes on SST-2 dataset, it surpasses the MLP, Bi-LSTM, and GCN models).

Among the models that only use local information, there is a significant improvement for BERT over MLP and Bi-LSTM. It shows that the attention model represented by BERT can easily surpass the DNNs model represented by MLP and the RNNs model represented by LSTM.

Between the models that exploit a vocabulary graph, VGCN can always outperform Text GCN, although not too much. This result is consistent with the results of the experiments of VGCN in the previous Section (Table A.2).

Comparing VGCN/Text-GCN and BERT, using only the vocabulary graph and the other using local context, we see that BERT can always outperform VGCN/Text-GCN, which shows that taking full advantage of document context itself improve the overall performance. However, GCNs only take into account the global graph information without local information such as word order.

The performance of Bi-LSTM using local information is between Text-GCN and VGCN. Although LSTM can use the information of sequence order and capture non-consecutive/long-distance information in the local sequence, it performs worse than BERT on word sense disambiguation [**66**]. Yet after convolving global language information, GCNs can eliminate the ambiguity or enhance the meaning that is not obvious in local information (see the visualization section below).

Models such as Vanilla-VGCN-BERT and Interactive-VGCN-BERT combining local and global information present a better result than the other baseline models. This result proves

Table 4.2. Weighted average F1-Score and (Macro F1-score) on the test sets. We run 5 times under the same preprocessing and random seed. Macro F1-score and Weighted F1-Score are the same on SST-2. Bold indicates the highest score and underline indicates the second highest score.

Model	**SST-2**	**CoLA**	**ArangoHate**	**FountaHate**
KNN	55.45	56.42 (47.62)	58.82 (56.28)	51.47 (41.82)
Naive Bayes	82.68	60.51 (49.46)	81.41 (80.96)	74.68 (56.07)
SVM	77.15	60.91 (50.07)	82.43 (82.01)	77.47 (62.60)
MLP (DNN)	80.78	61.39 (53.20)	84.71 (84.42)	79.22 (65.33)
Text-GCN	80.45	56.18 (52.30)	84.77 (84.43)	78.74 (64.54)
Bi-LSTM	81.32	62.88 (55.25)	84.92 (84.58)	79.04 (65.13)
VGCN	81.64	63.59 (54.82)	85.97 (85.69)	79.00 (64.04)
BERT	<u>91.49</u>	<u>81.22</u> (<u>77.02</u>)	87.99 (87.75)	80.59 (66.61)
Vanilla-VGCN-BERT	91.38	80.70 (76.30)	<u>88.01</u> (<u>87.79</u>)	<u>81.11</u> (<u>67.86</u>)
Interactive-VGCN-BERT	**91.93**	**83.68** (**80.46**)	**88.43** (**88.22**)	**81.26** (**68.45**)

the benefit of combining local information and global information. The Interactive-VGCN-BERT outperforms Vanilla-VGCN-BERT, for the interactions between local and global information. The superior performance of Interactive-VGCN-BERT clearly shows the benefit of allowing interactions between the two types of information.

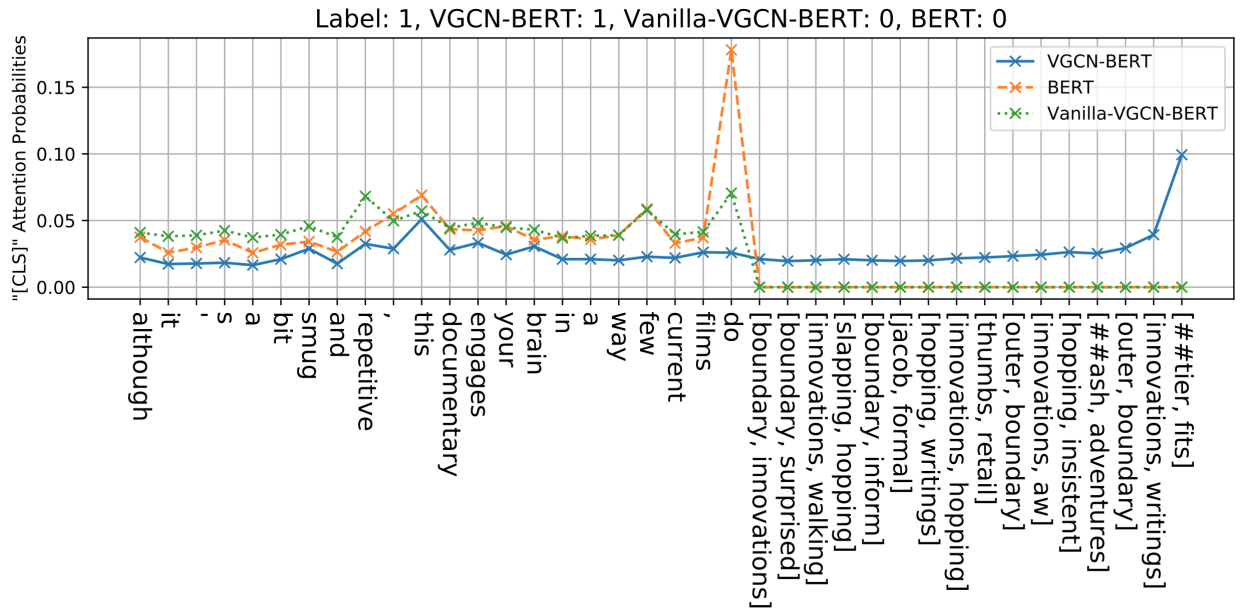
4.7. Visualization

To better understand the behaviors of BERT, and its combination with VGCN, we visualize the attention distribution of the [CLS] token in the self-attention module of BERT, VGCN-BERT and Vanilla-VGCN-BERT models. As the vocabulary graph is embedded into vectors of 16 dimensions, it is not obvious to show what meaning corresponds to each dimension. To facilitate our understanding, we show the top two words from the sub-graph related to the input sentence, which are strongly connected to each of the 16 dimensions of graph embedding. More specifically, when each word embedding of a document is input to Equation 3.2.2, we only need to broadcast the result of XA and element-multiply it by W to obtain the representation value of the words involved. The equation for obtaining the involved words' id is as follow:

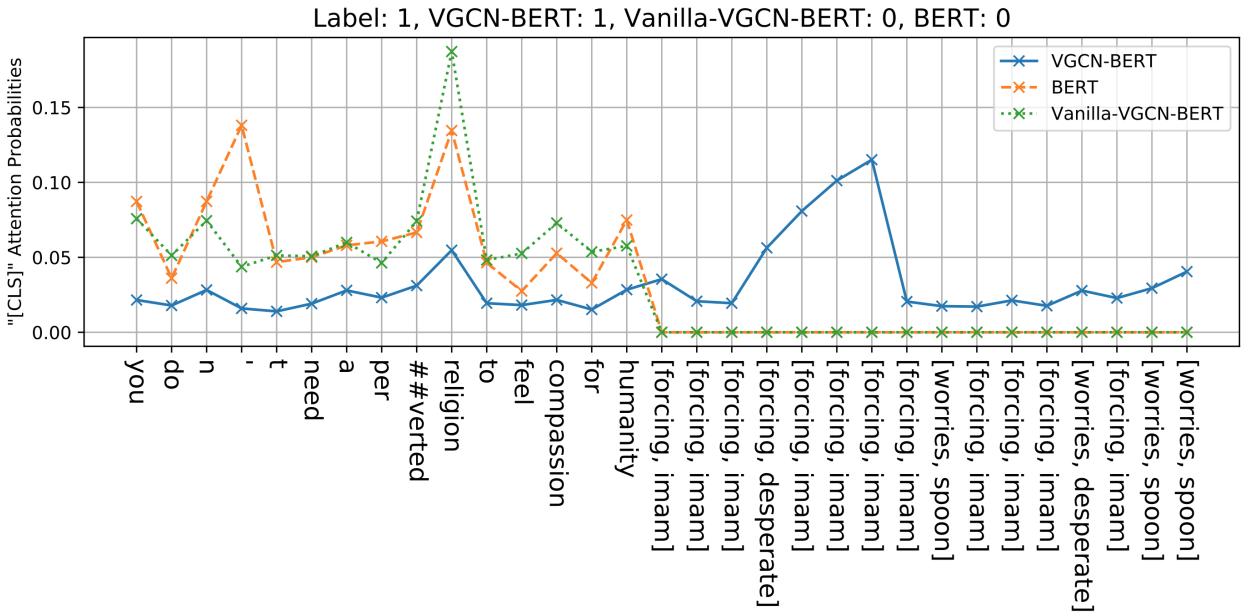
$$Z = (\vec{x}A)^T \odot W, \tag{4.7.1}$$

$$\text{IDs involved} = \arg\text{sort}(Z[:,g]), \tag{4.7.2}$$

where x is a document in row vector, $g \in [1,G]$, $G = 16$ is the size of graph embedding. For example, the first dimension of graph embedding shown in a) of Figure 4.1 corresponds roughly to the meaning of *"[boundary, innovations]"*.



(a) This example comes from *SST-2* dataset. Class 1 is positive.



(b) This example comes from *ArangoHate* dataset. Class 1 is offensive.

Figure 4.1. Visualization of the attention that the token [CLS] (used as sentence embedding) pays to other tokens. The first part corresponds to word embeddings of the sentence. The second part is the graph embedding. [word1, word2] indicates the approximate meaning of a dimension in graph embedding.

In Figure 4.1, we show the attention paid to each word (embedding) and each dimension of graph embedding (second part). As BERT does not use graph embedding, the attention corresponds to graph embedding is 0. In Interactive-VGCN-BERT, we see that graph embedding draws an important part of attention.

For example a) of the movie review: *"Although it's a bit smug and repetitive, this documentary engages your brain in a way few current films do."* The first half of the sentence is explicitly negative, while the remaining part expresses a positive attitude in an implicit way, which makes the sentence difficult to judge. For this example, BERT pays a very high attention to *"do"*, and a quite high attention to *"this"*. These words do not bear much meaning in sentiments. The final classification result by BERT is 0 (negative) while the true label is 1 (positive).

Vanilla-VGCN-BERT concatenates graph embedding with BERT without interaction between them. We can see that still no attention is paid to graph embedding, showing that such a simplistic combination cannot effectively leverage vocabulary information.

Finally, for Interactive-VGCN-BERT, we see that a considerable part of attention is paid to graph embedding. The graph embedding is produced by integrating gradually the local information in the sentence with the global information in the graph. In the end, several dimensions of the graph embedding imply the meaning of *"innovation"*, to which quite high attentions are paid. This results in classifying the sentence to the correct class (positive).

The meaning of *"innovation"* is not produced immediately, but after a certain number of layers in BERT. In fact, through the layers of BERT, local information in the input sentence is combined to generate a higher-level representation. In this example, at a certain layer, the expression *"a way few current films do"* is grouped and represented as an embedding similar to the meaning of *"innovation"*. From then, the meaning related to *"innovation"* in the graph embedding is capture through self-attention, and reinforced later on through interactions between the local and global information. It should be noted that although the attention weights in the figure for *"innovation"* in the figure are not significantly higher than other local words, but we see that the model actually selected these two words repeatedly in multiple graph embeddings, which means the cumulative impact is high enough.

For example b) of the offensive language: *"You don't need a perverted religion to feel compassion for humanity."* The sentence is a double negative sentence and is vague for a machine, further, *"perverted"* is not a common offensive word, and it rarely appears in the corpus. In general, the obscure meaning, combined with the new words that are not easy to train, lead to wrong judgment. In fact, the word *"perverted"* does not exist in the vocabulary of the pre-trained BERT, thus BERT split this new word into two words – *"per"* and *"##verted"*, which exist in the BERT vocabulary. This is a customary treatment for the new words in the NLP approaches. Yet this treatment method will cause the pre-trained word embedding to fail to fully express the meaning of the original word. Besides, the occurrence of *"perverted"* in this application-specific corpus (*ArangoHate* dataset) is only a few times, this results in that during the model fine-tuning phase, the model cannot effectively adjust the weights for this word. The final classification result by BERT is 0 (normal) while the true label is 1 (offensive).

Although Vanilla-VGCN-BERT concatenates graph embedding with BERT, it obtains not only *"forcing imam"*, but also other background words convolved by VGCN, such as *"worries spoon"* and so on. It has no mechanism to effectively interact between local words and global words, therefore it cannot highlight *"forcing imam"*, then such a simplistic combination cannot effectively leverage vocabulary information.

Finally, for Interactive-VGCN-BERT, we see that the most attention is paid to *"forcing imam"* in the graph embedding part, and the combination of these two words convolved has a clear meaning of infringing religion. After fully interacting through the layers of BERT, *"forcing imam"* is highly selected, and can be grouped with other local words (e.g. *"religion"*, which makes up for the impact of BERT's split *"perverted"*, and eventually results in classifying the sentence to the correct class (offensive).

Chapter 5

CONCLUSION

Hate speech is a growing problem on social media. Many users on social media are subject to hate speeches from other users. Despite the efforts of governments and social media platforms, the problem is not eradicated because the task is not trivial. In many cases, the words used in a post may not express a hateful thought, but the whole post is. The problem stems from the very complexity of natural language in which the expressions are very flexible and various. Better automatic detection tools are thus needed to prevent hate speech to be spread on social media. This thesis presents an investigation on this problem.

Hate language detection can be cast as a classification problem: given a post, we want to determine if it contains hate speech. This problem has been investigated in many existing studies. In general, the existing approaches rely on the content of the post, without a component that tries to capture the general knowledge about the language. That is, only the local information present in the post is used in these approaches. We believe that in many cases, a post contains hate speech because we make reference to the general language. Some posts may seem inoffensive if we analyze the words in the post alone, but it may be offensive if we put it into the general usage context of the language. Therefore, in this thesis, we proposed an approach that combines the local information from the post and the global information about the language. This latter part is captured in this thesis by the relations between words, which is a way to reflect some global knowledge about the language.

More specifically, we proposed a model that combines global background language information and local information, and allow them to interact through the process. The thesis contains two main parts. First of all, we explored a way to create a representation for a set of words connected in a graph. We proposed a general form of graph for the graph convolutional network – Vocabulary GCN. The motivation was that the existing GCN approaches all assume that the documents to be classified should be present in the graph when the training process is performed. This is unrealistic. We want to design an approach that is inductive, i.e. it can be applied to a new text. To do this, we map documents to the words they contain, and the graph - vocabulary graph - only contains words, which can be generally

applied to any text. The structure of our VGCN is simpler than traditional GCNs, and it can be trained in mini-batch mode, and can be used both in the case of citation networks and in the case of general text classification without citation network. It can be used as an upstream model for other models and provides the graph embedding for downstream models. In our experiments, we showed that it is not only superior in performance to Text GCN and Fast GCN - two representative GCN models for text classification, but also reduces the time complexity and the space complexity by several times or even dozens of times.

The second part of our study is on the integration of a graph embedding with the local information from the text. In our study, the latter part is captured by BERT. So, we designed an Interactive-VGCN-BERT model, in which the graph representation and the text representation of BERT can interact through the multi-layer multi-head self-attention mechanism to create a representation at higher layers. In our experiments, we show that Interactive-VGCN-BERT works better than another combination of VGCN and BER without interaction - Vanilla-VGCN-BERT.

Our experiments have tested various approaches to classify hate speeches, as well as other classification tasks (movie review and grammaticality): Text-GCN, Bi-LSTM, VGCN, BERT, and VGCN-BERT. The approach we proposed consistently produced the best effectiveness. In addition, it produced new state-of-the-art performance on two offensive language datasets (*ArangoHate* and *FountaHate*).

In this thesis, we used a basic vocabulary graph created using mutual information between words extracted from text collection. Many other lexical resources can be used to create an alternative graph. For example, Wordnet can form another type of vocabulary graph, which may be complementary to ours. Therefore, it would be interesting to investigate the utilization of such resources in the future.

Our approach to hate language detection (and to ext classification in general) only uses the text content. It is known that social media is a strongly connected network: users are connected, posts are also connected (post-reply-comments). The connected information in such a network can be very useful for hate language detection. For example, the previous behaviors of a user could be a good indicator whether the person is respectful. The meaning of the post (e.g. a comment) is also strongly related to the post it is connected to. These elements should be taken into account in hate language detection, which would be an interesting problem to study in the future.

References

[1] Sweta Agrawal et Amit Awekar : Deep learning for detecting cyberbullying across multiple social media platforms. *In Advances in Information Retrieval - 40th European Conference on IR Research, ECIR 2018, Grenoble, France, March 26-29, 2018, Proceedings*, pages 141–153, 2018.

[2] Aymé Arango, Jorge Perez et Barbara Poblete : *Transduction (machine learning)*. https://en.wikipedia.org/wiki/Transduction_(machine_learning).

[3] Aymé Arango, Jorge Perez et Barbara Poblete : *Hate Speech Detection is Not as Easy as You May Think: A Closer Look at Model Validation*. Paris, 2019.

[4] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta et Vasudeva Varma : Deep learning for hate speech detection in tweets. *In Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017*, pages 759–760, 2017.

[5] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner *et al.* : Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent et Christian Janvin : A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin et Tomas Mikolov : Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

[8] Gerlof Bouma : Normalized (pointwise) mutual information in collocation extraction. University of Potsdam, 2009. Proceedings of the Biennial GSCL Conference 2009.

[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam et Yann LeCun : Spectral networks and locally connected networks on graphs. *In 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[10] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt et Gaël Varoquaux : API design for machine learning software: experiences from the scikit-learn project. *In ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[11] Hongyun Cai, Vincent W Zheng et Kevin Chang : A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9): 1616–1637, 2018.

[12] Jie Chen, Tengfei Ma et Cao Xiao : Fastgcn: Fast learning with graph convolutional networks via importance sampling. *In ICLR*, 2018.

[13] Corinna Cortes et Vladimir Vapnik : Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[14] Thomas Davidson, Dana Warmsley, Michael Macy et Ingmar Weber : Automated hate speech detection and the problem of offensive language. *In Proceedings of the 11th International AAAI Conference on Web and Social Media*, ICWSM '17, pages 512–515, 2017.

[15] Michaël Defferrard, Xavier Bresson et Pierre Vandergheynst : Convolutional neural networks on graphs with fast localized spectral filtering. *In NIPS*, pages 3844–3852, 2016.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee et Kristina Toutanova : BERT: pre-training of deep bidirectional transformers for language understanding. *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[17] Antigoni-Maria Founta, Constantinos Djouvas, Despoina Chatzakou, Ilias Leontiadis, Jeremy Blackburn, Gianluca Stringhini, Athena Vakali, Michael Sirivianos et Nicolas Kourtellis : Large scale crowdsourcing and characterization of twitter abusive behavior. *In 11th International Conference on Web and Social Media, ICWSM 2018*. AAAI Press, 2018.

[18] Björn Gambäck et Utpal Kumar Sikdar : Using convolutional neural networks to classify hate-speech. *In Proceedings of the First Workshop on Abusive Language Online, ALW@ACL 2017, Vancouver, BC, Canada, August 4, 2017*, pages 85–90, 2017.

[19] Yoav Goldberg et Omer Levy : word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014.

[20] Alex Graves, Abdel rahman Mohamed et Geoffrey Hinton : Speech recognition with deep recurrent neural networks. *Acoustics, speech and signal processing (icassp)*, pages 6645–6649, 2013.

[21] Will Hamilton, Zhitao Ying et Jure Leskovec : Inductive representation learning on large graphs. *In NIPS*, pages 1024–1034, 2017.

[22] Mikael Henaff, Joan Bruna et Yann LeCun : Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[23] Sepp Hochreiter et Jürgen Schmidhuber : Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang et WANG Houfeng : Text level graph neural network for text classification. *In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3435–3441, 2019.

[25] Dziadek J, Henriksson A et Duneld M : Improving terminology mapping in clinical text with context-sensitive spelling correction. *Informatics for Health: Connected Citizen-Led Wellness and Population Health; IOS Press: Amsterdam, The Netherlands*, 235:241–245, 2017.

[26] Max Jaderberg, Karen Simonyan, Andrea Vedaldi et Andrew Zisserman : Reading text in the wild with convolutional neural networks. *International Journal of Computer Vision*, 116(1):1–20, 2016.

[27] Chanwoo Jeong, Sion Jang, Hyuna Shin, Eunjeong Park et Sungchul Choi : *A Context-Aware Citation Recommendation Model with BERT and Graph Convolutional Networks*. arXiv:1903.06464, 2019.

[28] Shengyi Jiang, Guansong Pang, Meiling Wu et Limin Kuang : An improved k-nearest-neighbor algorithm for text categorization. *Expert Syst. Appl.*, 39(1):1503–1509, 2012.

[29] Thorsten Joachims : Text categorization with support vector machines: Learning with many relevant features. *In Machine Learning: ECML-98, 10th European Conference on Machine Learning, Chemnitz, Germany, April 21-23, 1998, Proceedings*, pages 137–142, 1998.

[30] Rie Johnson et Tong Zhang : Effective use of word order for text categorization with convolutional neural networks. *In NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 103–112, 2015.

[31] Ian T. Jolliffe et Jorge Cadima : *Principal component analysis: a review and recent developments*. 2016.

[32] Karen Spärck Jones : A statistical interpretation of term specificity and its application in retrieval. 1972.

[33] Armand Joulin, Edouard Grave, Piotr Bojanowski et Tomas Mikolov : Bag of tricks for efficient text classification. *In EACL*, pages 427–431. Association for Computational Linguistics, April 2017.

[34] Yoon Kim : Convolutional neural networks for sentence classification. *In EMNLP*, pages 1746–1751, 2014.

[35] Yoon Kim : Convolutional neural networks for sentence classification. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014.

[36] Thomas N Kipf et Max Welling : Semi-supervised classification with graph convolutional networks. *In ICLR*, 2017.

[37] Tuomo Korenius, Jorma Laurikkala, Kalervo Järvelin et Martti Juhola : Stemming and lemmatization in the clustering of finnish text documents. *In Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 625–633, 2004.

[38] Kamran Kowsari, Donald E. Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S. Gerber et Laura E. Barnes : Hdltex: Hierarchical deep learning for text classification. *In 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, Cancun, Mexico, December 18-21, 2017*, pages 364–371, 2017.

[39] Yann LeCun, Yoshua Bengio et Geoffrey E. Hinton : Deep learning. *Nature*, 521(7553):436–444, 2015.

[40] Shang Lei : A feature selection method based on information gain and genetic algorithm. *2012 International Conference on Computer Science and Electronics Engineering (IEEE)*, 2012.

[41] Jake Lever, Martin Krzywinski et Naomi Altman : Classification evaluation. *Nature Methods*, 13(8):603–604, 2016.

[42] Qimai Li, Zhichao Han et Xiao-Ming Wu : Deeper insights into graph convolutional networks for semi-supervised learning. *In AAAI*, 2018.

[43] Yujia Li, Daniel Tarlow, Marc Brockschmidt et Richard S. Zemel : Gated graph sequence neural networks. *In 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[44] Pengfei Liu, Xipeng Qiu et Xuanjing Huang : Recurrent neural network for text classification with multi-task learning. *In IJCAI*, pages 2873–2879. AAAI Press, 2016.

[45] Zhibin Lu, Pan Du et Jian-Yun Nie : VGCN-BERT: augmenting BERT with graph embedding for text classification. *In* Joemon M. Jose, Emine Yilmaz, João Magalhães, Pablo Castells, Nicola Ferro, Mário J. Silva et Flávio Martins, éditeurs : *Advances in Information Retrieval - 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14-17, 2020, Proceedings, Part I*, volume 12035 de *Lecture Notes in Computer Science*, pages 369–382. Springer, 2020.

[46] Viny Christanti M., Rudy Rudy et Dali S. Naga : Fast and accurate spelling correction using trie and damerau-levenshtein distance bigram. *TELKOMNIKA Telecommunication, Computing, Electronics and Control*, 16:827–833., 2018.

[47] Laurens van der Maaten et Geoffrey Hinton : Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605, 2008.

[48] Christopher D. Manning, Prabhakar Raghavan et Hinrich Schütze : *Introduction to information retrieval*. Cambridge University Press, 2008.

[49] Viny Christanti Mawardi, Niko Susanto et Dali Santun Naga : Spelling correction for text documents in bahasa indonesia using finite state automata and levinshtein distance method. *The 3rd International Conference on Electrical Systems, Technology and Information (ICESTI 2017)*, 2018.

[50] Tomas Mikolov, Kai Chen, Greg Corrado et Jeffrey Dean : Efficient estimation of word representations in vector space. *In 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[51] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado et Jeff Dean : Distributed representations of words and phrases and their compositionality. *In NIPS*, pages 3111–3119, 2013.

[52] Vinod Nair et Geoffrey E. Hinton : Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010.

[53] Dabal Pedamonti : Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *CoRR*, abs/1804.02763, 2018.

[54] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song et Qiang Yang : Large-scale hierarchical text classification with recursively regularized deep graph-cnn. *In WWW*, pages 1063–1072, 2018.

[55] Jeffrey Pennington, Richard Socher et Christopher D. Manning : Glove: Global vectors for word representation. *In In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 14, pages 1532–1543., 2014.

[56] A. Rajaraman et J.D. Ullman : *Data Mining*. 2011.

[57] Xin Rong : word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[58] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner et Gabriele Monfardini : The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[59] Dominik Scherer, Andreas C. Müller et Sven Behnke : Evaluation of pooling operations in convolutional architectures for object recognition. *In Artificial Neural Networks - ICANN 2010 - 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III*, pages 92–101, 2010.

[60] Junyuan Shang, Tengfei Ma, Cao Xiao et Jimeng Sun : Pre-training of graph augmented transformers for medication recommendation. *In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5953–5959, 2019.

[61] Jasmeet Singh et Vishal Gupta : Text stemming: Approaches, applications, and challenges. *ACM Comput. Surv.*, 49(3):45:1–45:46, 2016.

[62] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng et Christopher Potts : Recursive deep models for semantic compositionality over a sentiment treebank. *In In Proceedings of the 2013 conference on empirical methods in natural language processing (EMNLP)*, volume 14, pages 1631–1642., 2013.

[63] Kristijan Spirovski, Evgenija Stevanoska, Andrea Kulakov, Zaneta Popeska et Goran Velinov : *Comparison of different model's performances in task of document classification*. 2018.

[64] Ilya Sutskever, James Martens et Geoffrey E. Hinton : Generating text with recurrent neural networks. *In Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1017–1024, 2011.

[65] Duyu Tang, Bing Qin et Ting Liu : Document modeling with gated recurrent neural network for sentiment classification. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1422–1432, 2015.

[66] Gongbo Tang, Mathias Müller, Annette Rios et Rico Sennrich : Why self-attention? A targeted evaluation of neural machine translation architectures. *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4263–4272, 2018.

[67] Ashish Vaswani et Noam Shazeer etc : *Attention Is All You Need*. Long Beach.

[68] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò et Yoshua Bengio : Graph attention networks. *In ICLR*, 2018.

[69] Ruishuang Wang, Zhao Li, Jian Cao, Tong Chen et Lei Wang : Convolutional recurrent neural networks for text classification. *In International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*, pages 1–6, 2019.

[70] Yequan Wang, Minlie Huang, Li Zhao *et al.* : Attention-based lstm for aspect-level sentiment classification. *In EMNLP*, pages 606–615, 2016.

[71] Alex Warstadt, Amanpreet Singh et Samuel R Bowman : Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

[72] Zeerak Waseem : *Are You a Racist or Am I Seeing Things? Annotator Influence on Hate Speech Detection on Twitter*, 2016.

[73] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu et Kilian Q. Weinberger : Simplifying graph convolutional networks. *In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 6861–6871, 2019.

[74] Keyulu Xu, Weihua Hu, Jure Leskovec et Stefanie Jegelka : *How Powerful are Graph Neural Networks*. arXiv:1810.00826, 2019.

[75] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola et Eduard Hovy : Hierarchical attention networks for document classification. *In NAACL*, pages 1480–1489, 2016.

[76] Liang Yao, Chengsheng Mao et Yuan Luo : Graph convolutional networks for text classification. *In AAAI*, 2019.

[77] Yue Zhang, Qi Liu et Linfeng Song : Sentence-state lstm for text representation. *In ACL*, pages 317–327, 2018.

Appendix A

Experiments on Full VGCN

Our full VGCN can work on documents that are either independent and connected in a graph. In the experiments presented in Chap. 4, we only reported the experiments on documents that are independent because the test collections on offensive language detection do not include connections between documents. In this appendix, we show the experiment on classification with VGCN for documents that are either connected or not. We compare our VGCN with other traditional GCNs in both cases. In both cases, we investigate the following research question: How effective is our method in text classification with or without citation networks compared with existing approaches?

A.1. Text Classification without Citation Networks

We assume that texts are independent. We use the same datasets as in [**76**], including 20-Newsgroups (20NG), Ohsumed, R52, and R8 of Reuters, and Movie Review (MR). The overview of the five datasets is depicted in Table A.1.

Baselines: We compare our methods with the following methods:

— **CNN** [**34**]: a CNN-based method;

— **LSTM** [**44**]: a LSTM-based method;

— **Graph-CNN** [**15**]: a CNN-based method with word embedding similarity graph;

— **Text-GCN** [**76**]: a GCN-based method with a global graph for all documents and words;

— **Text-GNN** [**24**]: a GNN-based method with a graph for each document. In this case, the author did not publish the source code, so we directly quote the performance in the paper;

— **Fast-GCN** [**12**]: a GCN-based method enhanced with importance sampling.

Experimental Settings: We set the hidden size of the convolutional layer among $\{100, 200, 250, 500, 550\}$, the dropout rate to 0.6, the learning rate among $\{0.016, 0.018, 0.02\}$, the window size among $\{10, 20, 25, 50\}$, the weight decay among $\{0, 5e-5\}$, the maximum

Table A.1. Summary statistics of five datasets [**76**]

Dataset	Docs	Train	Test	Words	Nodes	Classes	Average Len
20NG	18,846	11,314	7,532	42,757	42,757	20	221.26
R8	7,674	5,485	2,189	7,688	7,688	8	65.72
R52	9,100	6,532	2,568	8,892	8,892	52	69.82
Ohsumed	7,400	3,357	4,043	14,157	14,157	23	135.82
MR	10,662	7,108	3,554	18,764	18,764	2	20.39

training epochs with Adam to 800, the early stop epoch to 10. The parameter settings in all baseline models are the same as in [**76, 12**].

Performance: As shown in Table A.2, our proposed method achieves the best performance in all datasets, which demonstrates the effectiveness of our method. When comparing with Text-GCN, our method yields around 1% improvement in the accuracy. Recall that Text-GCN builds a large graph including documents (and test documents) and word nodes, while our model builds a graph containing only word nodes. The results show that our graph can generate comparable or slightly better document representations to those generated via Text-GCN, based on the word graph where no test documents are included.

Both Fast-GCN and Text-GCN do not consider global correlations between words, which can be captured in our graph. They are unable to utilize global word correlation information. The comparison between our method and these methods show the importance of capturing global word correlations in a graph.

Note that the results of the graph-based methods tend to outperform other non-graph based methods, such as CNN, LSTM, and fastText. The reason is that the latter cannot leverage the information from a graph structure, while the graph-based methods can learn more expressive representations for nodes considering their neighborhoods.

In Table A.3, We further use different order of A_{vv} on the datasets of R52 and R8 to test the accuracy of our method. We choose these two datasets because other datasets will cause out of GPU memory when calculating A_{vv}^2 and A_{vv}^3. On both datasets, our method achieves the best performance when the 1-order neighborhood information in A_{vv} is incorporated. With 0-order A_{vv}, our method degrades to a two-layer MLP model with no neighborhood information incorporated. On the other hand, using higher-order information may over-propagate information to moderately related documents and noise can be introduced.

Parameter Sensitivity: Figure A.1 and A.2 shows the accuracy with our model on five datasets using different hidden dimension of the convolutional layer and sliding window size in PMI calculation. We can see that: 1) On R8, R52, and Ohsumed, the test accuracy improves when the hidden dimension increases up to 250, then it drops slowly. 2) For R8, MR and 20NG datasets, varying hidden dimension has limited influence on the test accuracy. Overall, it shows that the hidden dimension can neither be too small to embed enough information for propagation in the graph, nor too large to focus on important information for classification.

Table A.2. Test Accuracy on document classification task for 10 times running and report mean ± standard deviation. VGCN with one convolutional layer steadily outperforms other methods on 20NG, MR, R52 and R8 based on student t-test ($p - value < 0.05$). **OOM:** Out of GPU Memory.

Model	20NG	MR	Ohsumed	R52	R8
CNN	0.8215±0.0052	0.7775±0.0007	0.5844±0.0106	0.8759±0.0048	0.9571±0.0052
LSTM	0.7318±0.0185	0.7768±0.0086	0.4927±0.0107	0.9054±0.0091	0.9631±0.0033
Graph-CNN	0.8142±0.0032	0.7722±0.0027	0.6386±0.0053	0.9275±0.0023	0.9699±0.0012
Fast-GCN	**OOM**	0.7510±0.0021	0.5441±0.0081	0.8515±0.0045	0.9538±0.0036
Text-GCN	0.8634±0.0009	0.7674±0.0020	0.6836±0.0056	0.9356±0.0018	0.9707±0.0010
Text-GNN[**24**]	-	-	0.6940±0.0060	0.9460±0.0030	0.9780±0.0020
VGCN	**0.8885**±0.0012	**0.7794**±0.0010	**0.6962**±0.0024	**0.9486**±0.0009	**0.9790**±0.0014

Table A.3. Test Accuracy on document classification task for 10 times running using different order of A_{vv}, i.e. $A_{vv}^k, k \in (0,1,2,3)$.

Model	R52	R8
A_{vv}^0	0.9172 ± 0.0018	0.9509 ± 0.0013
A_{vv}^1	**0.9486** ± 0.0009	**0.9790** ± 0.0014
A_{vv}^2	0.9062 ± 0.0012	0.9671 ± 0.0015
A_{vv}^3	0.7558 ± 0.0012	0.8995 ± 0.0018

Similarly, on R8, R52, Ohsumed and MR datasets, our methods yield the best result when the window size is around 20, while smaller or larger window size can capture insufficient or noisy information. For 20NG dataset, the test accuracy consistently increases with the growing of window size. A larger window size may help capturing sufficient word correlation information with long documents, yielding better performance.

The above experiments show that the hidden dimension and window size should be chosen according to each test collection. It may not be good to use the same setting for different datasets.

Analysis of NPMI Threshold: As described in Section 3.2.1, we also examine NPMI VGraph in preparation for the combination of VGCN-BERT later. To examine the impact of the NPMI on the VGCN model, we set the threshold of NPMI from 0-1 with an interval of 0.1 for a total of 11 thresholds.

Figure A.3 shows the performance change of the model with NPMI=1 as the baseline. We find that with the increase in the threshold of NPMI, the accuracies of the five databases show the same trend, their accuracies reach the highest near 0.2, and reach the lowest near 0.75. When the threshold of NPMI is equal to 1, the model degenerates to a model that does not use the vocabulary graph because at this time the adjacency matrix $A = I$, and

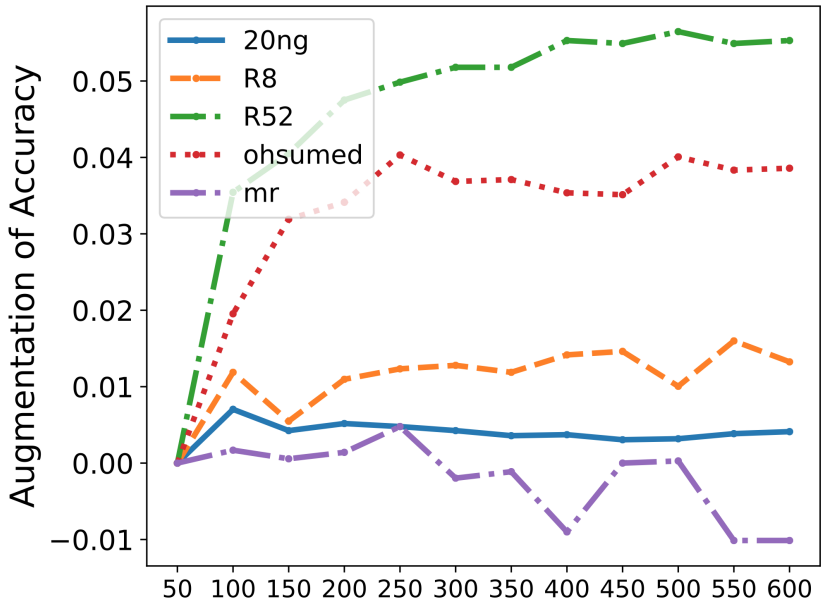


Figure A.1. Improvement of test accuracy with VGCN under different hidden dimension.

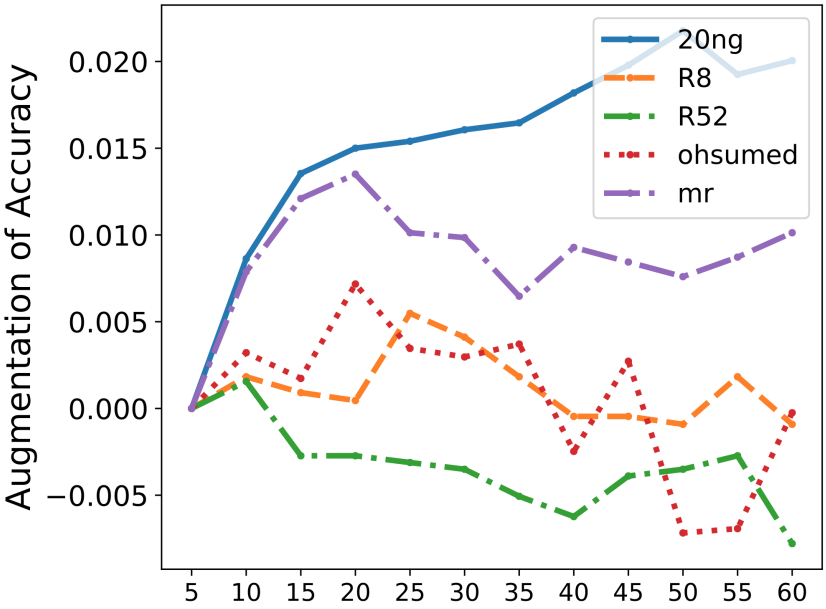


Figure A.2. Improvement of test accuracy with VGCN under different sliding window size.

the first layer of model XAW degenerates into XW. The higher accuracy in the range of $[0, 0.5]$ shows that the vocabulary graph can greatly help.

Efficiency: We further demonstrate that our method is significantly faster and has lower memory consumption compared with other GCN-based methods, with comparable and even better classification performance. We show in Table A.4 the time cost and the GPU memory cost of different models on one epoch of training. We can see that VGCN has

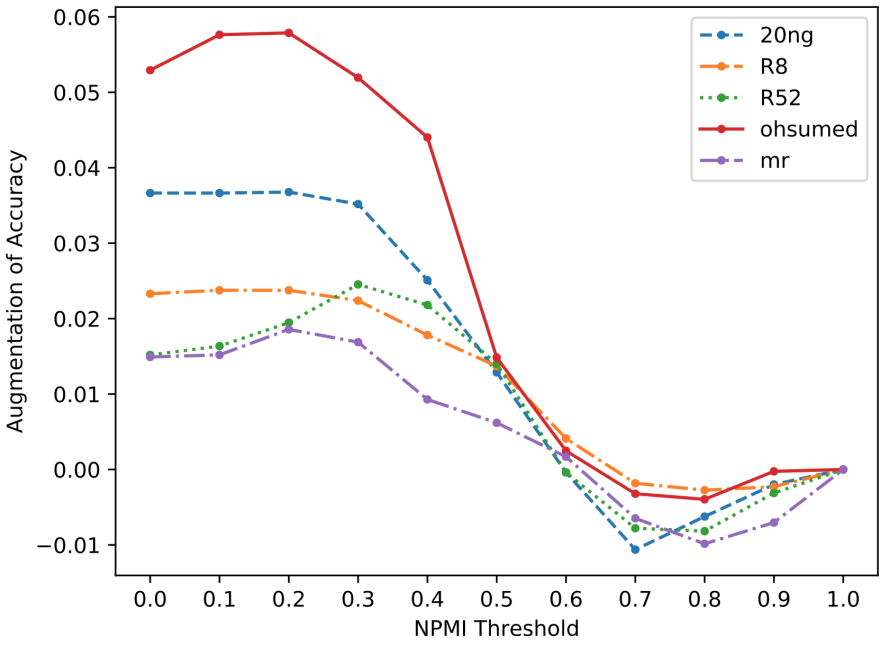


Figure A.3. Influence of NPMI threshold on VGCN performance. We use different NPMI thresholds (horizontal axes) to filter the edges of the vocabulary graph and obtain the corresponding accuracy, and then use the accuracy at the threshold of 1 as the benchmark to draw the growth of accuracy (vertical axis).

Table A.4. Time cost (ms) and GPU memory space cost (MiB) on one epoch of training under CUDA. (GPU: Nvidia Tesla K40c, CPU: Intel(R) 8 Core(TM) i7-2600K CPU @ 3.40GHz)

Model	**20NG**	**MR**	**Ohsumed**	**R52**	**R8**
	Time / Space	Time / Space	Time / Space	Time / Space	Time / Space
Fast-GCN	**N/A / OOM**	9,036 / 8,841	4,016 / 8,841	4,826 / 8,585	3,602 / 4,489
Text-GCN	**N/A / OOM**	1,060 / 3,995	4,303 / 2,960	2,365 / 2,119	1,869 / 1,613
VGCN	6,435 / 2,556	458 / 567	1,905 / 959	1,020 / 717	812 / 638

obvious advantages on both costs. In terms of time cost, Fast GCN is on average 14.6 times longer than VGCN, and Text-GCN is on average 5.1 times longer than VGCN. In terms of space cost, Fast GCN is on average 14.0 times larger than VGCN, and Text GCN is 4.5 times larger than VGCN. Notice that on the large dataset 20NG, Fast-GCN and Text-GCN models cannot be trained with GPU due to out of memory problem. Different from Fast-GCN and Text-GCN, our model removes useless nonlinear function and collapses multi-weight matrices into a single weight matrix among consecutive GCN layers. This significantly reduces both consumptions in time and memory.

Table A.5. Statistics of the citation network datasets [**73**]

Dataset	Nodes	Edges	Classes	Train/Dev/Test Nodes
Citeseer	3,327	4,732	6	120/500/1,000
Pubmed	19,717	44,338	3	60/500/1,000

A.2. Text classification with Citation Networks

In this subsection, we present and analyze the results on datasets with a predefined document-level graph, e.g, the citation network.

Datasets and Baselines: We use the same datasets as in [**36**], including Citeseer, and Pubmed citation datasets, in which a graph with the inter-document citation information is available. The overview of datasets is depicted in Table A.5. We compare our method with the following methods:

(1) **GCN** [**36**]: the original transductive GCN model,

(2) **Fast-GCN** [**12**]: an inductive GCN method which adopts importance sampling to do inference for unseen samples,

(3) **GIN** [**74**]: the Graph Isomorphism Network (GIN) which has a large discriminative power compared with GCNs,

(4) **SGC** [**73**]: a simplified linear GCN model.

Experimental Settings: We set the hidden size of the convolutional layer to 200, the dropout rate among $\{0.43, 0.5\}$, the learning rate among $\{0.0018, 0.018\}$, the weight decay among $\{0{,}8e-6\}$, the maximum training epoch with Adam to 1000 without early stopping. The parameter settings in all baseline models are the same as [**36, 73**].

Performance: As shown in Table A.6, our proposed method achieves the best performance over all baseline methods on Citeseer and Pubmed datasets. It shows that our model can be well generalized to test data using only document citation information in training data. Compared with baseline methods, besides the given citation network, our model also incorporates some additional inter-document correlation information derived from document-word features and word-word graph. This contributes to the better performance of our method.

In table A.7, we further present the test accuracy of our method with different orders of neighborhood information of A_{dd}. We can see that our method achieves the best performance when considering 1-order neighborhood information on both datasets. Considering 0-order citation information, i.e. the inter-document relationships are only calculated from document-word features, may not be sufficient for information propagation, while more than 1-order information may over-propagate to moderately related documents, resulting in worse performance in both scenarios.

This series of experiments show that our proposed VGCN can effectively solve several problems in the existing GCN approaches:

Table A.6. Test Accuracy (%) averaged over 10 runs on citation networks. VGCN steadily outperforms other methods on Citeseer and Pubmed based on student t-test ($p - value < 0.05$).

Model	**Citeseer**	**Pubmed**
GCN	70.3	79.0
Fast-GCN	68.8 ± 0.6	77.4 ± 0.3
GIN	70.9 ± 0.1	78.9 ± 0.1
SGC	71.9 ± 0.1	78.9 ± 0.0
VGCN	**72.2** ± 0.1	**79.6** ± 0.0

Table A.7. Test Accuracy (%) averaged over 10 runs on citation networks using different order neighborhood information of A_{dd}, i.e. $A_{dd}^k, k \in (0,1,2,3)$.

Model	**Citeseer**	**Pubmed**
A_{dd}^0	70.5 ± 0.0	78.0 ± 0.0
A_{dd}^1	**72.2** ± 0.1	**79.6** ± 0.0
A_{dd}^2	67.2 ± 0.1	78.9 ± 0.0
A_{dd}^3	64.8 ± 0.1	78.4 ± 0.0

1. It simplifies the graph to include only words, thereby reducing the complexity of the graph. As a result, the efficiency of convolution operations become much more efficient.

2. It transforms transductive learning to inductive learning. The documents to be classified do not need to be included in the graph when the model is trained. Therefore, VGCN is a model that corresponds better to the application situations.

The series of experiments are only on VGCN. It would be interesting to combine such VGCN with BERT. Unfortunately, this experiment cannot be done with our computation environment because the texts used in the VGCN tests are long. BERT is very computationally demanding. It does not work on long texts. Therefore, to apply our combined model VGCN-BERT to long texts, one has to find a way to encode long text efficiently. This will be an interesting future work.